www.java-pro.com
A Fawcette Technical Publication
August/September 1998, Vol. 2, No.4

JAVA Pro™

Making Connections with JDBC

Display Until Sept. 30, '98

INSIDE

Plugging into Oracle

Handling Events Like a Pro

Threading 101

Data Structures—from C to Java

REVIEWS

SuperCede for Java

eSuite DevPack

Data ExplorerJ 2.0 for JFC

Protospeed 2.0

COLUMNS

Ask the Java Pro
Fine Tuning the Socket and SocketImpl Classes

Javatecture
More on Design Patterns

Impure Thoughts

View Source

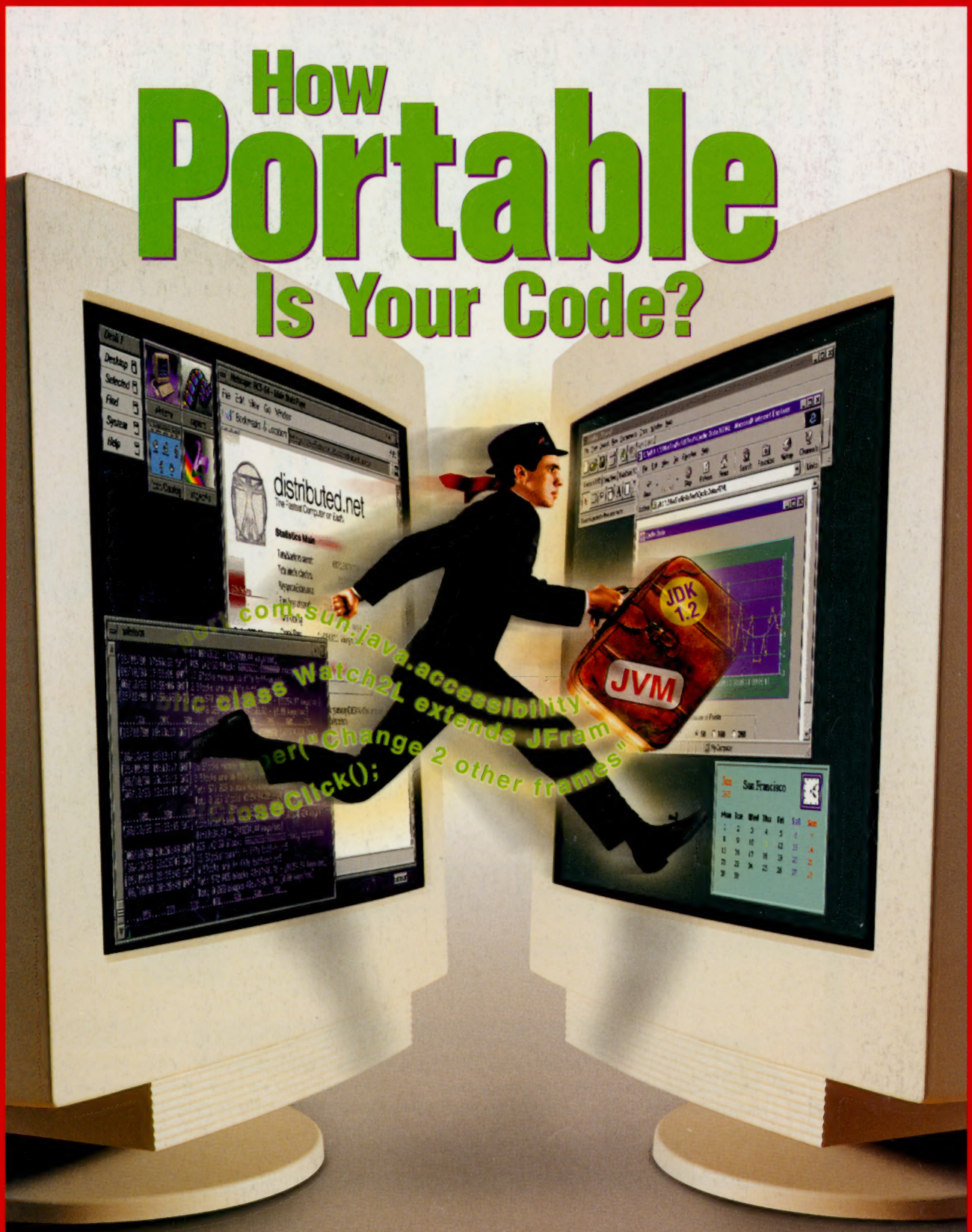
VM Roadtest
Benchmarking Microsoft's Jview JIT

U.S. $6.95 (Canada $7.95)

09>

0 74470 91607 9

How Portable Is Your Code?



REASONS TO USE JPROBE PROFILER:

1

A MILLION TIMES MORE ACCURATE THAN 'JAVA -PROF'

2

INSANELY EASY-TO-USE

JPROBE Profiler

Need more reasons to use JProbe to search and destroy the performance hot spots in your Java code? How about this: JProbe is the only Java profiler to accurately measure both time ***and memory*** consumption while your program runs. And it tracks everything for you, right back to each line of source code!*

If you develop Java code for a living, you need a professional Java profiler. So get a free evaluation copy of JProbe right now, and start making *your Java code* ***rock.***

Try JProbe Now! www.klg.com/jprobe

KL GROUP
Software Development Productivity

KL Group Inc. Toll-Free: 1-800-663-4723 Tel: 416-594-1026 Fax: 416-594-1919
KL Group Europe B.V. Tel: +31 (0)20 679 95 03 Fax: +31 (0)20 470 03 26
www.klg.com

* Source code not required to profile with JProbe. JProbe is a trademark of KL Group Inc. All other products are trademarks or registered trademarks of their respective companies. JP001

WANT THE LOWDOWN

ON CREATING KILLER

CUSTOM WEB APPS?

DOWNLOAD THIS.

esuite.lotus.com/devpackpreview

That's all it takes to get a free evaluation copy* of our Lotus eSuite™ DevPack™ (or, about $99 for a single-user edition gets you the whole magilla). And just what is eSuite DevPack? Only the most comprehensive set of kick-butt prebuilt, reusable, time-saving integrated Java™-based applets around. Use the sublime skills you already possess, plus eSuite DevPack, to build interactive Web apps that'll reduce the competition to chopped liver. 'Nuff said?

Lotus
Working Together®

*Evaluation copy offer good until September 1998. Restrictions may apply. For more information, call 1 800 872-3387, ext. E102. In Canada, call 1 800 GO LOTUS. © 1998 Lotus Development Corporation, an IBM Company. 55 Cambridge Parkway, Cambridge, MA 02142. All rights reserved. Lotus and Working Together are registered trademarks and eSuite and DevPack are trademarks of Lotus Development Corp. Java is a trademark of Sun Microsystems, Inc.

StudioJ

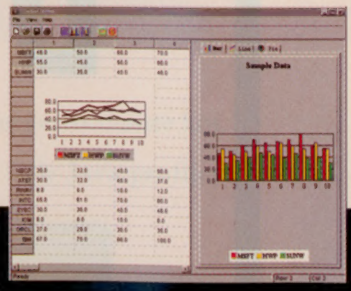
North American pricing
$995
Special upgrade pricing available for existing Rogue Wave and Stingray customers!

- Full source code included!
- No royalties or run-time fees!

a suite of Java tools

Integrated components

for developing great Java™ applications!



Why shop all over for Java components from different vendors and then spend days making them work together? **StudioJ** is your one-stop solution! Now you can get the high-quality, integrated, pure JavaBeans components and classes you need all in one place. To order or upgrade today call 1-800-487-3217 or visit:

www.roguewave.com/ad/studioj

Blend.J for adding over 45 AWT and JFC unique and rich extensions to your GUI front end.

Grid.J for table or spreadsheet layouts, with support for Excel-like formulas, binding to any data source, and much more.

Chart.J for dynamic, customizable charts, with data filters and overlays for composite charts.

DBTools.J for SQL portability across databases, with thread-safe and extensible classes.

Rogue Wave is a registered trademark, and StudioJ and .J are trademarks of Rogue Wave Software. Java and all Java-related trademarks are registered trademarks of Sun Microsystems, Inc. All other trademarks are the property of Rogue Wave Software or their respective holders.

The Software Parts Company™

Rogue Wave SOFTWARE

JAVA Pro™

August/September 1998,
Vol. 2, No. 4

Site Map

Features

18 Redefining Portability
by Tyler Sperry
Is Java doomed to a Write-Once, Test-Everywhere future? Tools such as Sun's Java Plug-in should help keep Java's stars uncrossed. Plus, Tyler gives us a rundown on the players in embedded and real-time Java.

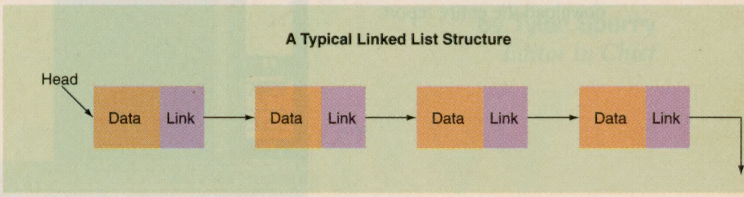
24 Making Connections with JDBC
by George Reese
JDBC helps Java deliver on the write once, run anywhere promise in database programming. Here's an overview to get you started.

32 Plugging into Oracle
by Dennis Harvey and Steve Beitler
Now that Oracle's had a Java Day, perhaps it's time to put Oracle to work in your Java applications.

42 Eventfully Speaking
by Larry O'Brien
Larry shows you how to use the Flyweight design pattern and other techniques to write programs that are flexible and fast.

48 Threading 101
by Luke Andrew Cassady-Dorion
Add multithreading to your next application for better response times.

56 From C to Java—a Structured Transition
by Peter Varhol
Your understanding of Java can be enhanced (or damaged) by your experience with older languages.



Departments

5 Editor's Note
by Tyler Sperry

7 Letters

12 New Products

77 Ad Index

Reviews

8 SuperCede for Java 2.01 Professional Edition
From its support for both JavaBeans and ActiveX to its full interoperability between Java code and C++ modules. Supercede for Java 2.01 is a first-rate development platform.

10 eSuite DevPack
Are your Web-based apps on the (Info)Bus? Lotus' eSuite DevPack lets you travel in style.

12 Data ExplorerJ 2.0 for JFC
ExplorerJ for JFC is a great tool for quickly building apps using the new Swing GUI components.

14 Protospeed 2.0
Progress Software's combination protocol debugger and distributed Java debugger will give you some useful insights into your apps, but version 2.0 is pretty rough around the edges.

Columns

60 Ask the Java Pro
by Daniel Savarese
Although much of the Java API's are well designed, there are some areas that need fine tuning to facilitate robust system programming. The Socket and SocketImpl classes constitute one such area.

64 Javatecture
by James Cooper
The Observer Design Pattern helps you keep data objects and display objects separated.

68 Impure Thoughts
by Andy Wilson
With Microsoft's latest release of Visual J++, building Windows applications that use the Common Object Model has gotten easier.

74 View Source
by Gary Cornell
Writing code for multiple browsers requires knowledge of both JavaScript and the underlying object models.

78 VM Roadtest
by Paul Tyma
Which Java sports car do you prefer: a Porsche or a Ferrari? We check out the latest VM offerings from Microsoft and Symantec.

Java Pro is published bi-monthly for $34.97 per year, by Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA, USA, 94301-2500. Tel. 650-833-7100; Fax. 650-853-0230. **Postmaster:** Send address changes to *Java Pro*, 209 Hamilton Avenue, Palo Alto, CA, USA, 94301-2500. **Customer Service:** For subscription orders, inquiries, or address changes, call 650-833-7100, send a fax to 650-321-3818, or write to *Java Pro*, 209 Hamilton Avenue, Palo Alto, CA, USA, 94301-2500. Foreign and Canadian orders must be payable in U.S. dollars, plus postage. The surface rate to Canada and Mexico is $40.97 per year and for all other countries the air mail rate is $56.97 per year. **Reprints:** For all *Java Pro* editorial and advertising reprints, please contact PARS International Corporation at 102 West 38th Street, 6th Floor, New York, NY 10018; call 212-221-9595; or fax 212-221-9195114. **Permissions:** If you would like to quote from an article, please call Monica McQueen at 650-833-7147, or mail the request to 209 Hamilton Avenue, Palo Alto, CA, USA, 94301-2500, Attn: Monica McQueen. Specify the issue date and title of the article, the portion you would like to quote, and the purpose. **Photocopy**

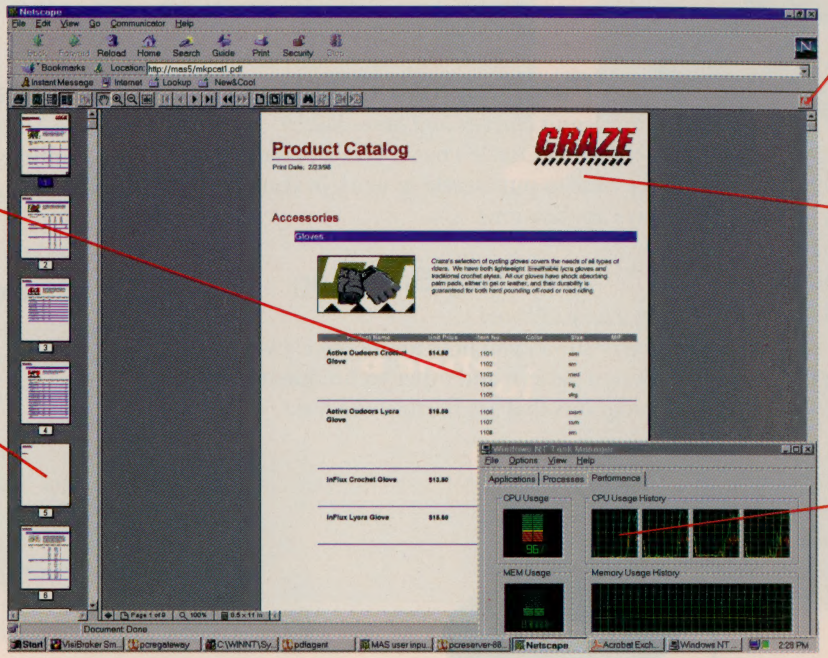
Rights: Permission to photocopy for internal or personal use may be granted by Fawcette Technical Publications. Please contact Monica McQueen for more information. © **1998 Fawcette Technical Publications**, all rights reserved. *Java Pro* is an independent publication not affiliated with Sun Microsystems. Sun Microsystems is not responsible in any way for the editorial policy or other contents of the publication. All contents of *Java Pro* are copyright © 1998 by Fawcette Technical Publications, unless otherwise noted. "VBITS" and "Interactive Developer" are trademarks of Fawcette Technical Publications, a California Corporation, James E. Fawcette, President. Java is a trademark of Sun Microsystems. Rather than put a trademark symbol in every occurrence of other trademarked names, we state that we are using the names only in an editorial fashion with no intention of infringement of the trademark. Although all reasonable attempts are made to ensure accuracy, the publisher does not assume any liability for errors or omissions anywhere in the publication. **Canadian GST#: 86831 0764 ISSN: 1096-4495**

How to quickly render long Web reports packed with charts and images.

Introducing new Parallel Crystal.

Can you print reports from your Java™ applets? **What you see is what you print** when you render report pages with Adobe® PDFs. Even the columns line up!

Download a single page without having to download the entire report.



Add **free Adobe readers** in all your browsers. After all, this is the Internet ... we charge by the server.

We bet that one of your colleagues already knows the Seagate Crystal® Reports APIs and Designer. Why learn new products for enterprise reporting when you can **leverage the skills you already have?**

Parallel processing with multiple Intel® P6 family processors FOCUSED ON SINGLE REPORTS renders your charts and images in record time.

You need the power of Parallel Crystal if ...

... you want scalability. Under our license from Seagate Software, Mobile Application Servers has modified the Crystal Report Engine (run-time) to harness the power of multiple processors in a single report. Now you have the power to quickly render long Web reports packed with charts and images.

... you want straight, simple, server-based pricing. There's no limit on the number of users per server and no per-client fees. Even technical support and upgrades are free for the first year. Choose from the following:

For software only, $10,000 per server. MAS or one of our integrator partners will come to your site anywhere in North America (and soon worldwide) for a day and install Parallel Crystal on your multiple processor server. Or, mail us your server and we will configure, test, and return it.

For approximately $25,000, we will send you Parallel Crystal pre-configured on a complete Intel-manufactured server with Dual Pentium® II processors, i960® RAID controller, and Seagate Cheetah™ hard drives.

For approximately $50,000, we will send you Parallel Crystal pre-configured on a complete Intel-manufactured server with Quad Pentium Pro processors, i960 RAID controller, and Seagate Cheetah hard drives.

For OEMs who need to embed Web Reports in applications for sale, we offer volume discounts, software manufacturing rights, UNIX ports, and custom distributed object interfaces.

... you want an interface that supports how you already work. Parallel Crystal uses a number of programming interfaces including the Visibroker™ implementation of the CORBA™ standard and Microsoft® ActiveX™.

Your Calling Application	Our Interface
Active Server Pages™	ActiveX or Java
Apple/Next WebObjects™	C
Browser	Java
Crystal Info™ APS	ActiveX or C or Java
NetDynamics™	PAC
NT® Terminal Server	ActiveX or C
Unix Applications	C
Visual Basic® 5	ActiveX

See for yourself. Download an applet, application, or page. You can even run Parallel Crystal remotely.

Go To: www.mobileapps.com

Mobile Application Servers, Inc.

320 West Port Plaza, Suite 205 • St. Louis, Missouri, 63146 USA • 314-205-8995 • Fax 314-205-8622 • 888-MOAPPS1 (toll free in USA) Business Partners/VAR inquiries welcome.

Seagate Crystal Reports is a registered trademark of Seagate Software, Inc. All other names and brands are the property of their respective owners.

Excess Baggage?

by Tyler Sperry
Editor in Chief

After years of faithful service, my trusty laptop finally called it quits. There were the usual warning signs: a hardening of the drive bearings and a whining noise that seemed to grow in volume with every looming deadline. Being somewhat untrusting by nature, I'd already backed up all my important files to tape or a server—but there's never a good time to lose a hard drive.

While I lament the loss of my little friend, the simple fact is, its replacement was inevitable. Less than three years after purchase, it's cheaper to buy a new laptop (superior in every respect) than to replace my old machine's disk drive. It's the nature of computer hardware.

As with hardware, so it is with software. You have to know when a comfortable, old friend has grown obsolete. Automatically upgrading to the latest, greatest technology is an expensive proposition. Too often, we find ourselves upgrading our tools because circumstances make it easy.

Claim checking. All of this brings me, in an admittedly circuitous fashion, to this month's cover story. I was first introduced to the question of virtual-machine portability while reading the traffic on the advanced Java mailing list. (See www.xcf.berkeley.edu/lists.html for the archives.) A developer was lamenting the apparently surreptitious replacement of one version of Microsoft's JVM with another. Having been burned by Usenet posts of dubious accuracy, I thought I'd check on this matter with one of our contributors. His response went like this: "Nonsense! I have both versions on my machine right now." There was the sound of keys clicking. "See, over here I have one version. And over here…" The sound of more keystrokes was followed by a disquieting silence, then a plaintive query: "How did *that* happen?"

The issue is not whether JVMs should be updated, of course. The issue is when and by whose authority the change should occur. Scheduling a server update when you've thoroughly tested your applications is one thing. Having an endless stream of support problems through variations in client engines is quite another.

Software engineering—if I can put those words together—is a balancing act. Most businesses don't want a technical buzz nearly as much as they want results. (There are companies that prefer new toys to working toys, but we usually refer to them in the past tense.) Working technology is essential, hence the importance of Sun's new Java Plug-in technology, where the developer can control both the version and timing of upgrades.

Upgrading Your Magazine. This engineering balancing act can show up in unexpected areas. For example, some shops have opted for a cautious approach to JDK upgrades. We have an article comparing the event handling models of JDK 1.0 and 1.1 in this issue, precisely because some of you may be wrestling with that transition. We will continue to mark our articles with version information so you'll know when an article relies on features found in a particular release of the JDK.

The latest technology is nice, but working technology is essential

But what about the rest of the magazine? As the newest Java magazine on the block, we've done our best to provide you with information to suit your current needs *and* let you plan for the future. If we're meeting that mark, let us know by sending feedback to java-pro@fawcette.com. Your comments let us know which features to drop and what improvements to make.

NEW
JAVA TECHNOLOGY
PREVIEW CENTER
Check it out!
http://cafe.symantec.com

VisualCafé™ for Java™

What the true professionals drive.

Fast. Flexible. And fully loaded.

Symantec's Visual Café for Java is the #1-selling Java development solution. New Version 2.5 combines state-of-the-art Java technology with the flexibility to work the way you want to. So you'll cross the finish line in a *fraction* of the time. Try it today!

FREE TEST-DRIVE!

Download a free trial version:
http://cafe.symantec.com

Or, call Corporate Sales
1-888-822-3409

The world's #1-selling Java IDE!

- Fastest Java compiler speeds your application development
- State-of-the-art technologies help race you through tedious debugging
- JIT 3.0 provides the fastest JIT performance available
- Customizable work environment, with intuitive user interface and macro-programmable editor, lets you work the way you want to
- Open environment lets you integrate your favorite tools, such as CASE and Source Code Management
- Native x86 compilation satisfies your fast run requirements
- 100% pure Java code ensures cross-platform portability

Symantec's Visual Café for Java is the most robust, productive, solution for building enterprise-ready applications with Java!

PC Magazine Editors' Choice · InfoWorld · JDJ World Class Award · Win 100 Magazine 1997 · c|net CNET.COM · 1996 Winner MVP PC Computing · Software Development Productivity Award 1997 · WEB Techniques Web Review Web Tools Awards · SYMANTEC. · Designed for Microsoft Windows 95

Symantec and the Symantec logo are U.S. registered trademarks and Symantec Visual Café is a trademark of Symantec Corporation. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems in the U.S. and other countries. Microsoft, Windows, the Windows logo and Windows NT are registered trademarks of Microsoft Corporation. Other brands and products are trademarks of their respective holder/s ©1998 Symantec Corporation. All rights reserved. In Canada, call 1-800-365-8641. In Australia, call 02-9850-1000. In Europe, call 31-71-535-3111.

JAVA Pro

Paul Spyksma, *Publisher*

Editorial

Tyler Sperry, *Editor in Chief*
John Dorsey, *Managing Editor*
Elise M. Peterson, *Associate Editor*

Contributors

Daniel Savarese, James W. Cooper, Andy Wilson
Gary Cornell, Paul Tyma, Michiel de Bruijn
Larry O'Brien, Peter Varhol, George Reese
Dennis Harvey, Luke Andrew Cassady-Dorian
Maureen Nelson, Steve Beitler

Art & Production

Michael Hollister, *Art & Production Director*
Amy Little, *Design Production Associate*
Christine Marquez, *Production Manager*
James Kattinge, *Senior Designer*
Terrilyn Chance, *Associate Art Director*
Kimberly Bell, *Electronic Products Associate Art Director*
Robyn Wandzell, *Electronic Production Associate*
Kim Le, *Senior Designer*

Advertising Sales

Brendan Coen, *Advertising Director*
Kevin Kirk, *Regional Manager*
Regan Swett, *Regional Sales Manager*
Monica McQueen, *Production Coordinator*
Lilly Gutierrez, *Recruitment Advertising Manager*

Circulation

Karen Koenen, *Circulation Director*
Katie Hathaway, *Associate Circulation Manager*
Patrick Bonstead, *Circulation Associate*
Todd Keffury, *Circulation Promotions Associate*
Paul Ceppaglia, *Circulation Coordinator*

Marketing

Sherry Thomas-Zon, *Marketing Director*
Julianne Olmstead, *Ancillary Products Manager*
Libby Dickinson, *Marketing Communications Manager*
Margaret Horoszko, *Marketing Designer*
Donna Ryan, *Marketing Specialist*

Conferences

Tena Carter, *General Manager*
Timothy Haight, *Editorial Director*
Ann ODonnell, *Conference Manager*
Daryl Haggard, *Conference Planner*
Janet Brandt, *Conference Planner*
Fred Perry, *Conference Planner*
Robert Scoble, *Conference Planner*
Damian Zusi, *Conference Sales Manager*

Electronic Products

Matthew Carter, *Editor & Producer*
Andrew King, *Associate Producer*
Christopher Kenton, *Project Manager*
Diane Matcheck, *Managing Editor*
Kay Keppler, *Senior Editor*
Michael Sturdivant, *Production Associate*
Sylvia Zele, *Associate Editor*
Adam Pavlacka, *Electronic Products Editorial Intern*

Electronic Products, Advertising

Ferris Ferdon, *Advertising Director*
Andrew Nickels, *Sales Support Manager*
Amanda Tyler, *East Coast Regional Manager*
Brendy Bell, *West Coast Regional Manager*

Customer Service

Douglas Smith, *Customer Service Manager*
Melissa Ramirez, *Customer Service Representative*
Chris Ortiz, *Customer Service Representative*
Noelle Ferdon, *Customer Service Intern*

Operations

Trisha Merriam, *Assistant to the Publisher*
Aaron Wuele, *IS Manager*
Jerry Sandefur, *Desktop Support*
Ron White, *Controller*
Connie Pulmano, *Facilities Manager*
Raylene Leavitt, *HR/Executive Administrator*
Stephanie Mattias, *Human Resources Assistant*
Daniel Hernandez, *Receptionist*

Fawcette Technical Publications

James E. Fawcette, *President*
Lynne Matthes, *Group Publisher*
Chris Kinsman, *Vice President, Technology*
Stephen Evanczuk, *Vice President, Electronic Products*
Wilson, Sonsini, Goodrich, & Rosati, *Corporate Counsel*

Letters to the Editor

Letters to Java Pro *are welcome. To be considered for publication, letters must include your name, address, and daytime phone number. Letters may be edited for form, fit, and style. Please send them to Letters to the Editor,* Java Pro, *c/o Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA 94301-2500; fax them to 650-853-0230; or e-mail them to java-pro@fawcette.com.*

Choosing the Right IDE

I'm glad I read your article ("IDEs–Which Route is for You?", April/May 1998), I learned a lot! I have to confess I am a beginning programmer and I didn't understand much of the technical stuff. For a beginning programmer who wants to learn Web based Java-database programming, which software package do you recommend? I thought that Visual Café by Symantec looked pretty good, but I haven't seen any real reviews on it yet. From your article, J++ also sounds very good. If you had to learn one of these from scratch, which one would you go with?

Pete Delalio
pete@top20.net

Editors respond: Your best bet is to explore some of the trial versions available online. This will let you explore several IDEs and discover which feels best to you. For Java database programming, there are many good products that support you in learning Java, JDBC, and will give you a working database engine to boot. Try visiting the Web sites of Inprise (formerly Borland), SuperCede, Sybase, and Symantec for starters. We reviewed Visual Café in our February/March 1998 issue. Also, check out the review of SuperCede for Java 2.01 in this issue. Unfortunately, database connectivity is one area where Microsoft's Visual J++ would not be our first choice.

Java is a blatent attempt to poison the Wintel hegemony

Look at your own www.devx.com poll. More than 80% of programmers reject [Microsoft's] blatant attempt to poison Java. But I'm less than surprised. Java itself is a blatant attempt to poison the Wintel hegemony.

However, I find my allegiance firmly on the side of Java (as distinct from Sun) and so take every opportunity to decry Microsoft's behavior.

Thanks for not blindly lashing out just because it had "Microsoft" written on the box, though. That was an unexpectedly refreshing change of pace for a Java publication.

Jon McGuire
jon@mcguire.net

Java Agents Online?

This article ["Java Mobile Agents," April/May 1998] was extraordinary. Unfortunately it was a second part of a two part article, how can I have access to the first article?

Christian Rauh
rauh@acm.org

Editors respond: All the articles and source code from our back issues are available online to Premier Club subscribers. Click on the "Register" button at our home page for more details.

Reviews

SuperCede for Java 2.01 Professional Edition

eSuite DevPack

Data ExplorerJ 2.0 for JFC

Protospeed 2.0

An Innovative Product Gets Better

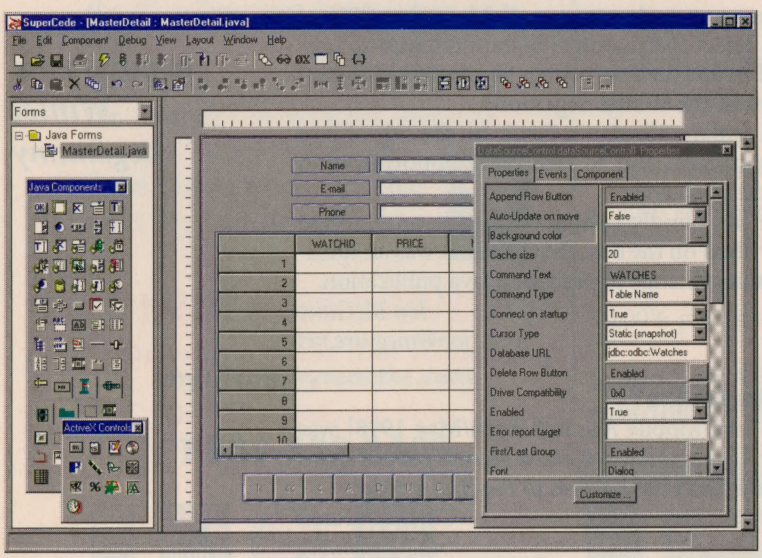
Quick Facts:

Although SuperCede 2.0 does not introduce as many new features as the original release, full JDK 1.1 support and improved database functionality make this a worthwhile upgrade.

by Michiel de Bruijn

When the first version of SuperCede was released in 1997, it brought some completely new features to the Java scene, such as native Intel code generation and the ability to change source code without having to recompile or even restart your application. Although now a number of other products (most notably Symantec's Visual Café) have matched this functionality, SuperCede has managed to produce a new version that, although not as dramatic as the original release, is still a major leap forward.

SuperCede for Java 2.01 Professional Edition
SuperCede Inc.
110 – 110th Avenue NE, Ste. 390
Bellevue, WA 98004-5840
Phone: 425-462-7242
800-365-8553
Fax: 425-637-5886
Web: www.supercede.com
Price: $995



Going 100% Pure. In addition to the ActiveX support included in earlier versions, SuperCede 2.0 allows you to include Java Beans on your forms. AWT layout managers also run at design time, preventing unanticipated changes in the look of your forms when you execute your code.

Most notably, SuperCede 2.0 provides complete support for version 1.1 of Sun's Java Development Kit (JDK), including Java Beans. To jump-start your application development, a collection of sixty Beans is included, and you can easily add third-party components by importing them into the SuperCede environment. However, like many other IDEs, the form editor only recognizes Beans that extend java.awt.Component. Layout managers now run in design mode as well, eliminating unwelcome changes to your form layout as soon as you run your project.

Support for Microsoft ActiveX components is still available, but now the focus is definitely on the Sun model. In the development environment, all components are referred to as "Beans", and although a pretty nice Bean Browser is provided, you're left to guess at the implementation of ActiveX controls yourself.

Strong Database Support

If you're not into database development, you're missing most of the fun in SuperCede 2.0. Both ODBC and JDBC connections are supported (a SuperCede-runtime-only version of the Intersolv database driver is included in the box), and creating data-aware applications is almost too easy. After connecting to a database, you can drag entire data tables or individual fields from the data sources browser onto your form, where they become data-aware controls.

In addition to automatically generating labels and navigational buttons for each field, SuperCede also allows you to easily generate master/detail forms. Setting up the relationship between the master and the detail part of the form (the latter typically being a grid control) requires no coding, provided your database has been designed properly.

The functionality of the included data-bound controls goes beyond the bare basics: In addition to the obligatory text, date/time, and currency controls, you get fully functional grid, image, and masked edit components.

Real Programmers Rejoice

In addition all that user friendliness, SuperCede still offers some unique features that will appeal to hard-core programmers. For example, the ability to compile C++ modules in addition to Java code and offer full interoperability between the two languages is unmatched by any competing product, and SuperCede's debugger still shines as an example of how such a thing should be implemented. The product also ships with a number of class libraries and third-party products that enhance the basic JDK functionality significantly.

A Few Rough Edges

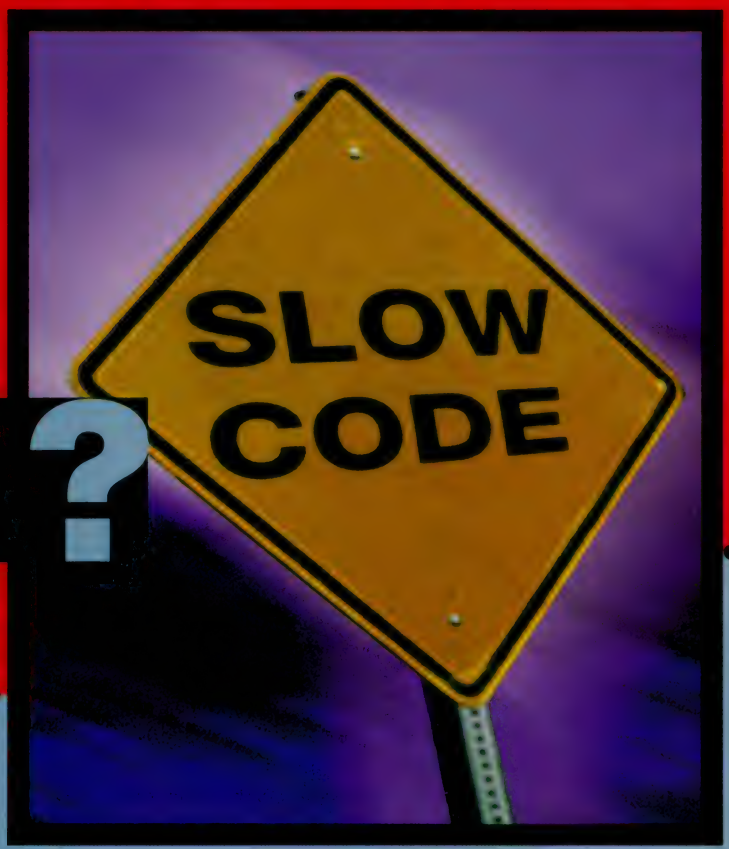
Unfortunately, SuperCede 2.0 has some rough edges. A service pack available from SuperCede's Web site (bumping the version number to 2.01) addresses a lot of the issues, but some major problems remain. For example, SuperCede still doesn't verify Java byte codes, making it roll over and die when confronted with bogus class files. Also, the advertised Visual Basic-to-Java conversion utility is not included in the box at this time, but the company promises that it will be available as soon as the incompatibilities between this third-party tool (supplied by TVObjects) and SuperCede's new form file format have been addressed.

These issues aside, SuperCede 2.0 is a solid product that will certainly appeal to its intended audience: application developers on the Wintel platform. ■

Michiel de Bruijn is a native of Rotterdam, The Netherlands, but currently lives in Cape Town, South Africa. He seems to enjoy living on the bleeding edge of technology. Michiel always welcomes questions and comments. Reach him at mdb@x42.net.

WARNING!

BUT WHERE?



ONLY NUMEGA TRUETIME™ PROVIDES SEAMLESS MULTI-LANGUAGE PERFORMANCE ANALYSIS FOR COMPONENT-BASED APPLICATIONS.

Compuware® NuMega™

Now, for the first time, you can easily and accurately pinpoint code performance bottlenecks no matter what language they are written in or where they reside. In Visual Basic, Visual C++, and Java, NuMega TrueTime lets you see exactly where problems occur. Get NuMega TrueTime and get all your applications and components up to speed.

1-888-686-3438 www.numega.com

Call to request a **FREE 14 DAY LICENSE.**

SmartDebugging™ for faster development VISUAL BASIC® VISUAL C++® JAVA™

30-DAY MONEY BACK GUARANTEE

Copyright©1998, Compuware Corporation. All rights reserved. All trademarks and registered trademarks are the property of their respective owners.

Building Blocks For Real-World Applications

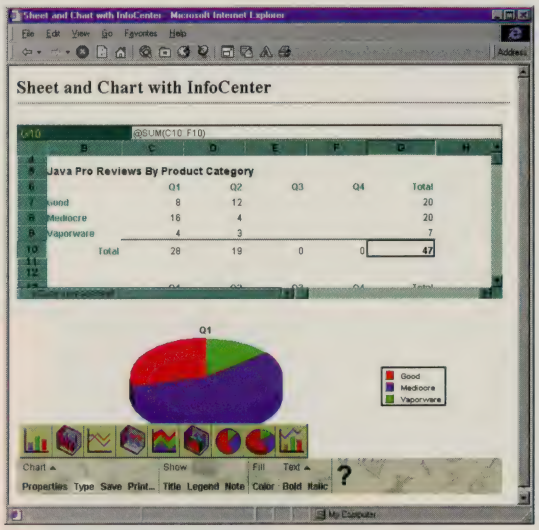
Quick Facts:

Although improvements in functionality and performance are still needed, this first version of Lotus eSuite DevPack is a good foundation for your Java-based productivity apps.

by Michiel de Bruijn

With the release of Lotus eSuite WorkPlace and eSuite DevPack, those who complain of the lack of "real" productivity applications written in Java will have to look for a new pet peeve. These products bring an impressive suite of business applications to any platform capable of running JDK 1.1-based applets. WorkPlace is intended for end-users, and provides a ready-to-run Java desktop. Lotus eSuite DevPack, reviewed here, is a developer product, making individual applets available for the rapid deployment of your Web-based business applications.

eSuite DevPack
Lotus Development Corp.
55 Cambridge Parkway
Cambridge, MA 02142
Phone: 617-577-8500
800-555-5921
Web: http://esuite.lotus.com
Price: $99 (single user); $1495 (single processor server)



Quickly Build Custom Productivity Apps. The eSuite DevPack spreadsheet and graph modules, shown running here in Internet Explorer 4.01 and scripted using HTML APPLET tags, automatically exchange information with each other and the "InfoCenter" menu bar using Lotus' InfoBus technology, which will also be part of a future Sun JDK release.

Got Disk Space?

DevPack is a large application. The complete installation, including all documentation and samples, takes up about 40 MB and consists of over 7,000 files. The JAR files containing the applets, which eventually get downloaded to your clients, total 7.5 MB. Installing one or more of the 21 international language packs available will add about 1 MB per locale.

Although Lotus currently only provides explicit support for running the DevPack applets on the latest Wintel versions of the Microsoft or Netscape browsers, I was successful in running them on other platforms. The only notable exception here were the two Network Computer (NC) devices that I tried, but since these also crash and burn when confronted with other "advanced" functionality (such as large animated GIFs), Lotus most likely isn't the party to blame here. Performance on the Wintel browsers was generally okay: Although the applet load time is noticeable, your applications should be sufficiently responsive once up and running.

Take Your Office Anywhere

With the initial release of the DevPack, Lotus includes spreadsheet, word processor, project scheduler, chart, and presentation graphics applets, as well as utility classes for functions like JDBC database access, file operations, and a CGI gateway. Unfortunately, the eSuite WorkPlace e-mail applets were left out of the DevPack, meaning you'll have to resort to third-party applications to build a complete end-user application.

The individual applets contain most of the functionality your end-users expect, but not everything. The word processor, for example, doesn't offer a spell checker and the spreadsheet supports only basic @ functions. The default format for most components is HTML/XML, with import and export support for the most popular file formats, including RTF for the word processor, WK1 for the spreadsheet, and MPP (Microsoft Project) for the scheduler applet.

Primarily A Web Tool

Although it is technically possible to use the DevPack applets from within a Java application—after all, they're regular Java Beans—the components included with this version are *really* intended to be embedded within and scripted from a browser environment. Lotus is planning on better integration with Java IDEs for future versions, but for now, it's best to stick to a general-purpose Web design tool, APPLET tags, and JavaScript for generating DevPack apps.

One of the nicest things about the DevPack is its InfoBus technology, which allows applets to exchange data amongst themselves by "publishing" data onto this bus without requiring any specific coding. For example, if you embed a menu/toolbar applet along with a spreadsheet component, this "InfoCenter" will automatically pick up the spreadsheet's current configuration from and send any user commands to it over the InfoBus.

Even with its current reliance on a browser environment and rather minimal functionality in some areas, the eSuite DevPack is an exciting demonstration of the capabilities of Java. Anyone with an interest in developing cross-platform productivity applications should take a look at the product (a free trial version is available from the Lotus Web site). Although enhancements such as better support for Java development environments are needed to make the DevPack more attractive to developers, the current release enables developers to build some pretty spectacular Web-based custom applications. ■

Michiel de Bruijn is a native of Rotterdam, The Netherlands, but currently lives in Cape Town, South Africa. He seems to enjoy living on the bleeding edge of technology. Michiel always welcomes questions and comments. Reach him at mdb@x42.net.

ObjectSpace

VoyagerPro. Pretty cool.

Easy to use.
Multilingual.

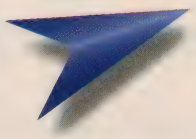
Innovative mobility.
Dynamic aggregation.

What the hell are we talking about?
See us at SD 98 East, Booth 220

Or call us

Or e-mail us

Or whatever.



VOYAGERPRO™

ObjectSpace
14850 Quorum Drive 5th Floor
Dallas, Texas 75240
Telephone 972 726 4100
Fax 972 726 4200
www.objectspace.com

Get Your Data In Full Swing

Quick Facts:

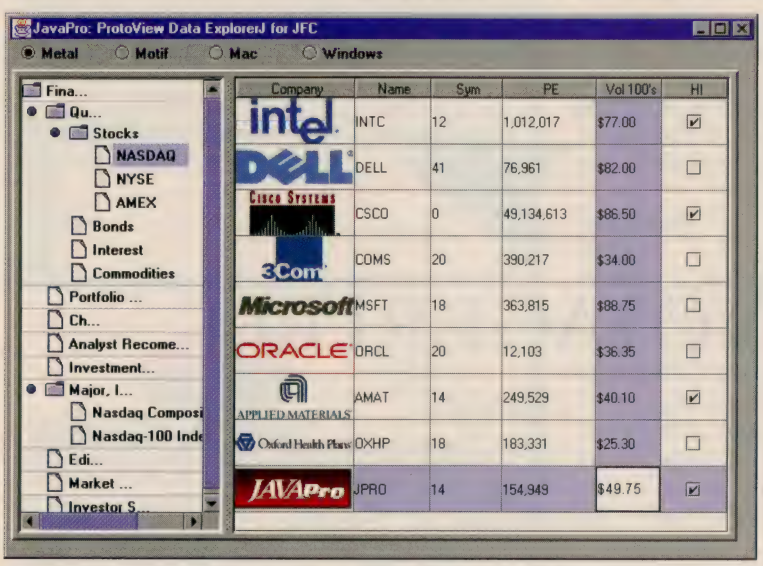
ProtoView's Data ExplorerJ for JFC makes good use of the new JFC "Swing" GUI components to allow rapid deployment of almost any application that needs to display structured data.

by Michiel de Bruijn

Data ExplorerJ for JFC
ProtoView Development Corp.
2540 Route 130
Cranbury, NJ 08512
Phone: 609-655-5000
800-231-8588
Fax: 609-655-5353
Web: www.protoview.com
E-mail: info@protoview.com
Price: $299 ($499 with subscription service)

Sun's Swing component set, the GUI toolkit that ships as part of the Java Foundation Classes (JFC), gives you the ability to create exciting interfaces that can adapt to the look and feel of various platforms. But, although Swing will save you a lot of work, getting the individual components to work together in a real application is not a trivial issue. That's where a tool like ProtoView's Data ExplorerJ for JFC comes in. I reviewed a late beta copy of this product, which is derived from ProtoView's popular ActiveX control, Data Explorer.

Data ExplorerJ is based on the JFC JTree, JSplitter, and JTable controls and automatically creates a treeview that allows users to drill down on data tables whose data content is displayed in a grid on the right-hand side of the control. The component also handles other basic functionality, such as searching, sorting, in-place editing, and data- type validation.



Select Your Favorite Look. Data ExplorerJ uses the ability of JFC components to adapt the look and feel of various platforms, so users will not be confronted with a "foreign" user interface. Apart from this visual gimmickry, the product offers some *real* functionality, and allows you to dramatically reduce the time needed to deploy good-looking data-based apps.

Bring Your Own Data

One thing that Data ExplorerJ does *not* provide is automatic data binding. Instead, you're responsible for loading data into the control yourself. For some purposes this might be a disadvantage, on the other hand it means the component is more versatile. If you want to load data from a non-JDBC data source, or even statically, you don't have to resort to hacks.

You can dynamically add data to the control by creating new instances of the ExNode class, which extends the normal Swing tree node. In addition to the actual data, an ExNode instance holds all necessary configuration information to display the data as well, and has methods for searching and sorting. Depending on the type of data in the ExNode, the ExTable class (derived from a regular Swing JTable) will handle the actual display and handling of insert/delete operations.

If you don't like the behavior of the built-in JTable data model or ProtoView's enhanced Data Explorer model (which incorporates explicit support for the treeview), you can provide your own code to implement this functionality. This provides unmatched flexibility, but is quite a bit of work—fortunately, the built-in models will do fine for most situations.

Customize It All

Once the data is loaded into the control you can sit back and relax. Data ExplorerJ automatically handles most end-user interactions with your control, including synchronization between the treeview and the data panel, and automatic sorting of columns (which is persistent even if new nodes are added). Search functionality is easy to implement as well, although this requires some extra user interface code.

Almost every visual aspect of the displayed data can be modified. In addition to choosing between the various GUI styles (such as Motif, Mac, or Windows) supported by the JFC, you can set the colors and highlights of each node, background, and focus color selections. It's also easy to include images with your data, and you have full sizing control as well.

A Rare Component

In a nutshell, Data ExplorerJ is one of those rare components that is attractive from a user interface and a component model/language point of view. If you want a practical example of the power of JFC, download a fully functional evaluation copy from the ProtoView Web site (www.protoview.com). Full versions are also available from the Web site, and in addition to the byte-code version of the control, ProtoView offers the full source code. ■

Michiel de Bruijn is a native of Rotterdam, The Netherlands, but currently lives in Cape Town, South Africa. He seems to enjoy living on the bleeding edge of technology. Michiel always welcomes questions and comments. Reach him at mdb@x42.net.

Training Videos for Internet Software™

KeyStone Learning Systems Corp.

Microsoft® Certified Solution Provider

Java 1.1

VIDEO TRAINING SET

Brian Maso
Professional Trainer
& Java Instructor

Level 1 Video

1.0 Installing the Java Developers Kit
1.1 CD-ROM Installation
1.2 Classpath Environment Variable
1.3 Using your Favorite Text Editor
2.0 Programming Fundamentals
2.1 Integer
2.2 Floating Point Data Types: Character and Boolean
2.3 Constants
2.4 Overview of Operators
2.5 Standard Arithmetic Operators
2.6 Binary Arithmetic Operators
2.7 relational Operators
2.8 Strings
2.9 Arrays
2.10 Array of String Objects
3.0 Flow Control
3.1 Overview
3.2 Conditional Statements
3.3 Determinate Loops (For)
3.4 Intermediate (While, Do)
3.5 Switch Statement (Break)

$89.95
Time: 1 Hour, 40 Minutes

Level 2 Video

1.0 Methods
1.1 Overview
1.2 Creating Methods
1.3 Shortcut for Recalling Methods
1.4 Scope of Variables
1.5 Receiving Information from Command Line
1.6 Recursion
1.7 Blocks
2.0 Understanding OOP
2.1 What is an Object
2.2 Anatomy of an Object
2.3 Building a Class
2.4 Assessor & Mutator
2.5 Tostring () Method
2.6 Bike 3 Example
2.7 Object Relationships
2.8 Uses a Relationship
3.0 Using Java Objects
3.1 Strings
3.2 Constructors & Assessors
3.3 String Tokenizer
3.4 Date Object
4.0 Utilizing the Benefits of OOP
4.1 Overview
4.2 Inheritance
4.3 Getfullyear
4.4 Runtime Information

$89.95
Time: 1 Hour, 49 Minutes

Level 3 Video

1.0 Abstract Windows Tool Kit
1.1 Overview
1.2 Creating a Blank Window (Frames)
1.3 Placing Text on the Window
1.4 Drawing Rectangles and Ovals
1.5 Draw Round Rectangles
2.0 Using Existing Java Applications
2.1 HTML Code
2.2 Animator Applet
2.3 Marquee
3.0 Graphical User Interface
3.1 File Dialog Box
3.2 Custom User Interface
3.3 Custom Dialog Box
3.4 Fonts
3.5 Color
3.6 Event Handling
3.7 Text Fields
3.8 Text Area
3.9 Displaying a Graphic Image File
3.10 Menu Structures

$89.95
Time: 2 Hours, 18 Minutes

Level 4 Video

1.0 Layout Managers
1.1 Grid Layout
1.2 Card Layout
1.3 Grid Bag Layout
2.0 Advanced User Interface
2.1 Choice
2.2 Anchor
2.3 List
2.4 Checkbox
2.5 Radio Buttons
3.0 Exception Handling
3.1 How to Handle Exceptions
3.2 Triggering your own Exceptions (Throw)
3.3 How Exceptions Propagate
3.4 Debugging your Exceptions (Print Stack Trace)

$89.95
Time: 1 Hour, 58 Minutes

Level 5 Video

1.0 File Input & Output
1.1 What is a File Input & Output
1.2 Creating a Java App that Performs I/O
1.3 Reading a File
1.4 Writing a File
2.0 Network Input & Output
2.1 What is Network I/O?
2.2 What is Client/Server Networking
2.3 How an HTTP Server Works
2.4 Building a HTTP Server

$89.95
Time: 2 Hours, 1 Minute

Level 6 Video

1.0 Threads
1.1 What is a Thread?
1.2 Building your first Multithreaded Program
1.3 Stopping a Thread
1.4 Building a Multithreaded HTPP Server
2.0 Applets
2.1 What is an Applet
2.2 Building your first Applet
2.3 Building a Virtual Tour Applet
3.0 Structured Query Language
3.1 What is SQL
3.2 Reviewing SQL in Microsoft Access
3.3 Creating new Records (Insert)
3.4 Retrieving Records (Select)
3.5 Modifying Records (Up-date)
3.6 Removing Records (Deleting
4.0 Open Database Connectivity (ODBC)
4.1 What is ODBC?
4.2 Installing ODBC
4.3 Configuring a Data Source
5.0 Java Database Connectivity (JDBC)
5.1 Installing Classes
5.2 SQL + JDBC + ODBC = Database Front End
5.3 SQL Select
5.4 SQL Update
5.5 SQL Delete
6.0 Common Gateway Interface (CGI)
6.1 Review of HTML Forms

$89.95
Time: 2 Hours, 7 Minutes

Special All Six $449.95

David Harding
Microsoft Certified
& Visual InterDev Instructor

Visual InterDev
Level 1/2/3 (ea) $89.95
Level 4/5/6 (ea) $89.95
Special-All Six $449.95

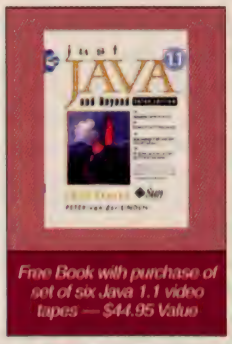
Visual J++ 1.1
Level 1/2/3 (ea) $89.95
Level 4/5/6 (ea) $89.95
Special-All Six $449.95

IIS 4.x
Level 1 $89.95
Level 2 $89.95
Level 3 $89.95
Special-All Three $249.95

HTML
HTML Lev 1/2/3(ea) $89.95
HTML Lev 4/5/6(ea) $89.95
DHTML Lev 1/2/3(ea) $89.95
Special-All Nine $649.95

Windows NT 4.0*
Server Admn. Lev 1 $89.95
Server Admn. Lev 2 $89.95
Covers Exam 70-67
Enterp. Admn. Lev 1 $89.95
Enterp. Admn. Lev 2 $89.95
Covers Exam 70-68
Serv. & Wrk. Install $89.95
Covers Exam 70-63 & 70-73
Workstn. Admn. Lev 1 $89.95
Workstn. Admn. Lev 2 $89.95
Workstn. Admn. Lev 3 $89.95
Covers Exam 70-73
Adv Level 1-3 (ea) $89.95
Workstn. User Level 1 $49.95
Workstn. User Level 2 $49.95
Special All Thirteen $859.95

Average: 2 hours of training per video



Free Book with purchase of set of six Java 1.1 video tapes — $44.95 Value

Ask about KeyNotes $14.95 ea
KeyStone's Quick Reference

Domestic Shipping: $5.00 for the first two video tapes; plus $1.50 for each additional video tape. International Orders: Shipping cost will vary, call for details.

All Videos Now Available in PAL (European Format)

RISK FREE 30 DAY Money Back Guarantee

4 Easy Ways to Order:

CALL ▶ 1-800-581-8258
801-375-8680

FAX ▶ 1-801-373-6872

WEB ▶ www.klscorp.com/jpm

MAIL ▶ KeyStone Learning Systems Corp.
2241 Larsen Parkway
Provo, UT 84606

AD# JP88

...are trademarks and/or registered trademarks of Novell, Inc., Microsoft Corp., Lotus Development Corp., Borland International Inc., and Aldus Corp., respectively

We accept C.O.D., Company P.O. or check +

Tapping into Distributed Applications

Quick Facts:

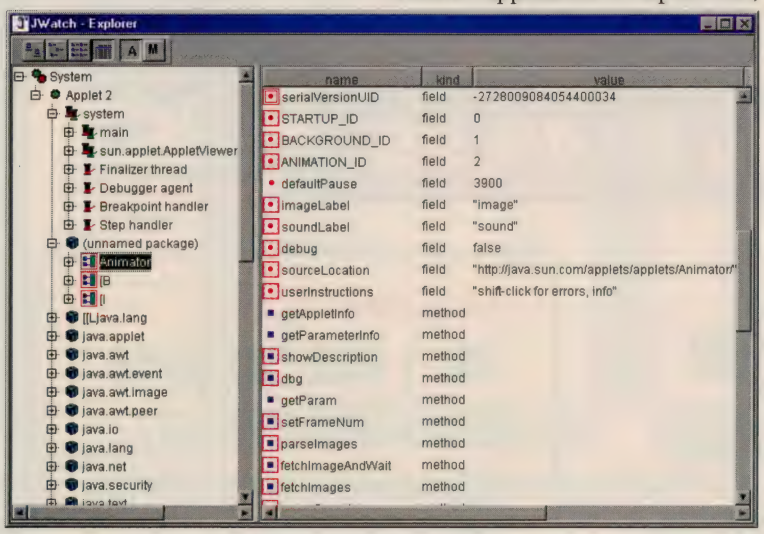
ProtoSpeed is an application bundle for testing and debugging intra- and Internet applications. Despite its 2.0 version number, however, it's not quite ready for prime time.

by Michiel de Bruijn

ProtoSpeed 2.0
Progress Software Corp.
14 Oak Park
Bedford, MA 01730
Phone: 781-280-4000
Web: http://protospeed.progress.com
Price: $495 (Introductory price, regularly $995)
Requires: Microsoft Windows 95 or Windows NT 4.0, a display driver with at least 256-color capability, Sun JDK 1.1.4 or later for distributed Java debugger and extensibility features

The distributed applications running on the Internet and on corporate intra- and extranets have caused interesting problems not only for developers, but also for those who test and monitor such apps. Progress ProtoSpeed 2.0 was especially designed to help with end-to-end application debugging, whether source code is available or not.

ProtoSpeed consists of two separate applications: a protocol debugger that allows you to take a peek at the interactions between hosts using the most popular Internet application-level protocols, and JWatch, a distributed Java debugger for dissecting applications and applets running in a variety of VMs. Although both debuggers are written in Java, Progress currently only supports running them on the Microsoft Windows platform, with a version for Sun Solaris in beta at press time.



Through the Looking Glass. ProtoSpeed's JWatch component allows you to see what's inside a Java class file and catch its exceptions, even if you only have the compiled version of the applet. If source code is available, you can step through it and set breakpoints where desired.

Looking Inside the Wire

ProtoSpeed's protocol debugger component can be configured as a proxy server for TCP/IP services based on the HTTP, FTP, SMTP, POP3 or IMAP4 protocols (unfortunately, LDAP, a nightmare to troubleshoot using conventional tools, is not supported). The IP address and port number of the actual service are configured on startup, and you subsequently have to configure your client software to communicate with the proxy instead.

The debugger then presents a treeview of the session data, intelligently indicating errors and displaying statistics, depending on the protocol in use. You can expand and reduce tree nodes to get the detail you want, and a breakpoint facility allows you to pause the session and display a hex/ASCII dump of the current protocol packet if a particular event occurs. Because the breakpoint settings are protocol-specific as well, doing things like interrupting a SMTP session whenever a recipient is rejected is much easier than using a general-purpose protocol analyzer. On the other hand, not all protocol extensions are supported, and there are no facilities to automatically decode authentication handshakes.

Although the protocol debugger generally works well, it has some shortcomings that limit its usefulness. Getting the application up and running, for example, can be quite a challenge. I had to move my JDK installation to a directory without spaces in its name before ProtoSpeed would run. Also, the memory footprint of each monitoring session (over 9 MB) as well as ProtoSpeed's sluggishness and need to modify the client setup, make it doubtful that you could successfully use this tool in any other environment than a low-throughput test setup.

Checking Out Your Applets

The second ProtoSpeed component, JWatch, is designed to load Java applications and applets, regardless of their source location, and display as much information about them as possible. You can load your bytecodes into a new instance of either the Sun or Microsoft VM, or attach to an application that's already running inside a remote Sun VM.

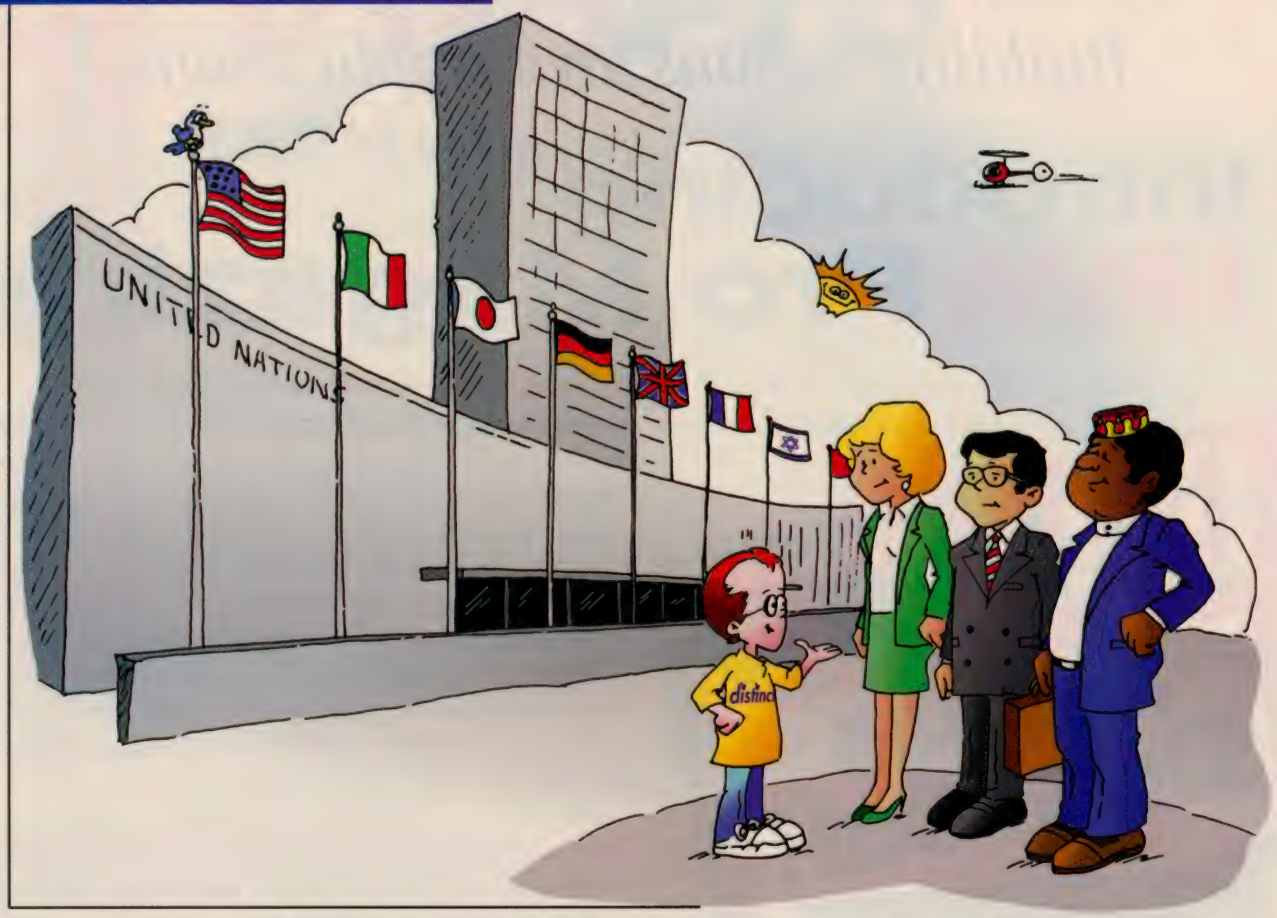
Once an application is loaded, you can view its details, such as the methods it implements, the packages its imports, and its local variables, in the Program Explorer. You can set breakpoints that are activated when a particular method is invoked, and if any exceptions are thrown during run time, you can catch them and view additional information about them as well.

As with the protocol debugger, the usability of JWatch is severely limited by its implementation. JWatch's performance was very slow, and during my limited evaluation of the product, I encountered several crashes as well as a number of applets that could not be processed due to their size or complexity.

Although ProtoSpeed is an interesting package, with some unique features, it needs major usability improvements before it will have anything to recommend it over traditional network packet analyzers or Java debuggers. ■

Michiel de Bruijn is a native of Rotterdam, The Netherlands, but currently lives in Cape Town, South Africa. He seems to enjoy living on the bleeding edge of technology. Michiel always welcomes questions and comments. Reach him at mdb@x42.net.

Doing it right with The Cody Kid.™



"Your heterogenous environments are a lot like ours."

Develop, debug and deploy cross-platform network applications fast, with Distinct RPC Toolkits.

Introducing the Distinct ONC RPC/XDR Toolkits, the fastest way to bring your corporate applications to the Windows 95 and NT desktop. In addition to our toolkit for C and C++ developers, Distinct delivers the first Java-based RPC toolkit. Distinct ONC RPC/XDR Toolkit for Java puts RPC-based distributed programming right at Java programmers' fingertips. And because ours is the first toolkit of its kind to receive 100% pure Java certification, you can now write platform-independent distributed applications. So, whether you are porting the front end of your UNIX applications to the Windows platforms, or creating a completely new distributed application over a heterogeneous network, rely on Distinct RPC toolkits to help you do it faster and more reliably. It's also the easiest way ever devised to bring your UNIX-based applications to your desktop.

Our RPC Toolkits are backed by comprehensive technical support from Distinct, a pioneer in enabling technologies for software professionals since 1984. To get your free trial copy, visit the Cody Kid™ on our Web site today.

distinct
Develop. Debug. Deploy.™

Download a trial version now at www.distinct.com or call 408-366-8933

DISTINCT is a registered trademark and the DISTINCT logo, product names, the DEVELOP. DEBUG. DEPLOY. tagline, DOING IT RIGHT WITH THE CODY KID., and THE CODY KID are trademarks of Distinct Corporation. Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc., in the United States and other countries. Windows and Windows NT are either trademarks or registered trademarks of Microsoft Corporation. All other trademarks are the property of their respective owners. ©1998 Distinct Corporation. 003

Finally, a Magazine for Programmers Building Serious Java Applications

Introducing JAVA™ Pro Magazine

Introducing *Java Pro*, the first independent technical magazine dedicated to serious Java development.

Java Pro is *the* professional developer's independent source for information on building mission-critical Java applications. *Java Pro* takes you beyond the hype to deliver hands-on, code-intensive, product-oriented information to help you harness Java's unique capabilities in your enterprise-wide development efforts. *Java Pro* tackles tough development issues and provides unbiased, no-nonsense coverage of the latest tools on the market.

Here's what you can expect from *Java Pro*:

- Independent comparisons of Java Rapid Application Development environments
- Tips and tricks for creating your first Java applications
- Extensive third-party product reviews and new product announcements
- Strategies to maximize productivity with re-usable components

Become a Charter Subscriber and Save over 28%!

Call today to subscribe to *Java Pro*. As a charter subscriber, you'll pay the low rate of $29.95 for a full year–six issues in all. That's a savings of more than 28% off the newsstand rate.

Money-back guarantee

If you are not satisfied with **JAVA Pro** at any time, ***we will refund the cost of your subscription.*** No questions asked.

To subscribe, call:

1.800.848.5523

Outside the U.S. call 650.833.7100
or fax 650.321.3818
Order online: http://www.devx.com

FTP FAWCETTE TECHNICAL PUBLICATIONS
209 Hamilton Avenue
Palo Alto, CA 94301-2500

Java is a trademark of Sun Microsystems, Inc.

One year is six issues. Savings is based on annual cover price of $41.70 for six issues. Subscription begins upon receipt of payment. Please allow 4-6 weeks for delivery of first issue. International subscriptions must be payable in U.S. dollars plus postage. Canada/Mexico add $9/year; all other countries add $22/year for airmail. Canadian GST included.

New Products

Utilities
Classes & Libraries
Developer Tools
Application Servers

Utilities

InstallAnywhere 2

Zero G Software's InstallAnywhere 2 is an installer authoring tool designed to allow Java application developers to deploy both Java and platform-specific software to multiple platforms. InstallAnywhere 2 will install a Java virtual machine if one is not available, set platform-specific options, and create double-clickable icons for Java applications so they appear as native applications. The InstallAnywhere 2 family includes three products; InstallAnywhere Standard Edition for mid-range developers, InstallAnywhere Express for small, independent developers, and Install Anywhere Commercial Edition for large commercial software and enterprise MIS developers. InstallAnywhere 2 Standard Edition is priced at $995.

Zero G Software
Phone: 415-512-7771
Fax: 415-512-7775
Web: www.ZeroG.com

Classes & Libraries

Interstream iIntegrale 1.0.4

Interstream iIntegrale is a package of controls to enhance the look and usability of Java applets and applications. The new release includes features such as Tree, Wizard, and Spreadsheet controls, and improved versions of the basic Java controls. iIntegrale controls inherit the background image of their iIntegrale container and have the ability to contain an image, allowing for a more graphic-oriented interface. iIntegrale has been tested for use with Microsoft Internet Explorer 3.0, 3.01, and 3.02, Netscape Navigator (for Windows 95) 3.0 and 3.02, and the "appletviewer" in the JDK 1.02. iIntegrale is available for $79.95, and a Source Code Edition is available for $399.95.

InterStream Inc.
Phone: 512-263-5469
Fax: 512-263-5440
E-mail: sales@i-stream.com
Web: www.i-stream.com

StudioJ

StudioJ is a new suite of JavaBean components and classes that includes full source code, which gives developers the ability to get under the hood and add customized extensions. StudioJ offers Java technology developers a complete solution for GUI development, data analysis, charting, and database access. It includes four libraries: Blend.J, Grid.J, Chart.J, and DBTools.J. Features include a set of over 45 JFC and AWT GUI controls, OO encapsulations for relational database development, a grid control with spreadsheet functionality, a set of charting Beans featuring 2D, 2-1/2D, and 3D chart types, overlay charts and data filters, and integration support between the four libraries. Priced at $995.

Rogue Wave Software
Phone: 303-473-9118
888-442-9641
Fax: 303-447-2568
Web: www.roguewave.com

Developer Tools

CoffeeShop

CoffeeShop (formerly known as SFS Software's iavaDraw) is a development platform for Java applet, applications, servlets, or JavaBean components. CoffeeShop is cross-platform compatible and is specifically designed to support the programmers' main working process: developing, debugging, and executing. Other features include an intelligent source code editor that is able to color code and remember nearly 2800 Java-related keywords; a WYSIWYG Graphical User Interface builder; with support for Layout Managers and JFC (Swing components), 20 predefined visual components, a graphical debugger interface, and project management with support for multiple projects. The product is available now through shareware distribution and is priced at $99 for the standard single user license. Academic Single user licenses are also available at $59. A trial version is available at www.sfs-software.com.

SFS Software
Phone: 49-172-471-4485
Fax: 49-3681-394015
E-mail: info@sfs-software.com
Web: www.sfs-software.com

JBuilder 2

Inprise Corporation (formerly Borland International Inc.) has released Borland JBuilder 2, a visual development tool for creating Java business and database applications for the enterprise. The product's open environment supports JDK 1.1.x, JDK 1.2, JFC/Swing components, JavaBeans, Enterprise JavaBeans, CORBA, RMI, and JDBC. JBuilder is available in three versions; JBuilder 2 Client Server/Suite, JBuilder 2 Professional, and JBuilder 2 Standard. JBuilder 2 Client/Server Suite includes support for multiple Java Development Kits, application deployment and management, Enterprise JavaBeans, Java Servlets, JFC/Swing components, CORBA, and high-productivity coding Wizards. JBuilder Client/Server Suite is priced at $2,495, JBuilder Professional at $799.95, and JBuilder Standard $99.95.

Inprise Corp.
Phone: 800-233-2444
Web: www.inprise.com/jbuilder

JProbe Profiler

JProbe Profiler is a graphical hierarchical profiler that enables Java developers to conduct code analysis on Java applets and applications. Designed specifically for Java, it allows developers to analyze a Java application's architecture and performance, and determine what modifications can be made to save time or reduce the consumption of memory resources. Some of the performance problems JProbe can highlight include inefficient algorithms, I/O blockage, excessive method calling, excessive thread creation, inefficient memory usage, and unwanted memory usage. JProbe is priced at $499, and a preview is available at www.klg.com.

KL Group Inc.
Phone: 416-594-1026
800-663-4723
Fax: 416-594-1919
Web: www.klg.com

Application Servers

Apptivity 2.1

Progress Software's Apptivity 2.1 is an application server and development tool for Java. Apptivity 2.1 incorporates a number of new features including HTTP tunneling for improved deployment of Web applications, an expanded database driver set, and enhanced quality and scalability. Priced at $1,995, it includes Apptivity Developer and a five-user Apptivity Application Server. Additional Apptivity Application Server license pricing begins at $5,000 for a 20-user server. Apptivity Application Server supports any Java server platform. Apptivity applications can be deployed to any 1.02 or 1.1 JDK platform, including Netscape Navigator 3.x/4.x and Microsoft Internet Explorer 3.x. Apptivity is compatible with any JDBC- or ODBC-compliant data source.

Progress Software
Phone: 781-280-4000
800-477-6473
Fax: 781-280-4901
Web: www.progress.com

Redefining Portability

Is Java doomed to a write-once, test-everywhere future?

From the beginning, Java has been cursed with hyperbole. "Write once, run anywhere" sounds appealing, but detractors have been quick to note any failure to deliver on that promise. Some have gone so far as to claim that cross-platform portability is a fool's errand, an impossibility. Computer systems are too diverse for any interpreter to bridge the gaps, they say. And besides, even if you should achieve platform independence, the performance of an interpreter is unacceptably slow. Does any of this sound familiar?

Tyler Sperry is Editor in Chief for *Java Pro* magazine. When not reading or writing about Java, he searches for other ways to get into trouble. Reach him at tsperry@fawcette.com.

Embedded Pretenders

Although Sun has provided a variety of Java APIs aimed at embedded systems, the company has steadfastly refused the concept of subsetting the Java language. (Unless you consider PersonalJava, EmbeddedJava, and JavaCard to be subsets of Java, that is.) Faced with a per-CPU licensing fee and an unsatisfactory Java specification, a number of companies have sought to create a leaner, meaner Java.

The two recurring themes in the products listed here are a need for a smaller memory footprint than the normal JRE and some means of addressing real-time performance. The original JRE required approximately 4 MB of RAM and ROM, which is nothing for desktop PCs but quite an issue for machines originally designed with under a megabyte of memory.

Garbage collection is a particularly challenging subject, as the "official" approach to garbage collection is non-deterministic. Not being able to predict the bounds of execution delays is unacceptable in real-time applications. Accordingly, some vendors have attempted to improve upon the original JVM design by changing the garbage collection scheme and/or dedicating a thread to background memory reclamation.

Hewlett-Packard. While HP's "embedded VM" has gotten a lot of press, most of it has been devoted to relatively non-technical issues, such as Microsoft's purchase of a license for the WinCE operating system. But Microsoft isn't the only company with a license; other companies have signed on, lured by both lower royalties and the promise of smaller memory demands (500 KB of ROM and less than 200 KB of RAM). HP's design is aided by the company's long history of success in developing and selling products based on embedded systems, the most notable example being printers. For more details, see www.hpconnect.com/embeddedvm/.

NewMonics. This company's approach, PERC, has been to create a "dialect" of Java (JDK 1.1) for real-time applications, using a custom compiler technology. See www.newmonics.com for details, including several articles and presentations on the design.

Negev Software Industries. This Israeli-based company developed the Software Co-Processor for Java (JSCP) as a family of products. Central to the design is the idea of encapsulating the Java-related activities while still providing access to the services of the underlying real-time operating system. Dynamic allocation of resources makes characterizing memory requirements difficult. See www.nsi.co.il/jscp_products.htm for more information.

In addition to products from commercial vendors, there are several experimental freeware projects. They support a larger range of processors, but don't provide commercial-level technical support.

Sidebar continued on next page.

Do these arguments have any merit? Well, yes. If you interpret "anywhere" strictly, WORA is an exercise in hyperbole. If you want to run your enterprise resource-planning suite on a TRS-80, you're going to have problems, regardless of your implementation language. But what about running your enterprise software on different servers? That's a different matter. For many developers supporting customers with different browsers isn't an impossible dream, but an everyday requirement. If you are one of the fortunate few who develop for a single platform, count your blessings now and feel free to skip to the next article. For those of you who need some level of cross-platform compatibility, however, there are some recent developments worth exploring. We might have to redefine "anywhere" to mean "any computer system used in our business," but that's a compromise most developers can live with.

Eliminating the Noise

If you've read much of the popular press about Java, you might be tempted to despair at the notion of portability. Bad news is what gets the ink, and it seems every week brings another story of lawsuits, feuding vendors, and virtual machine incompatibilities. Even tested and certified 100% Pure Java code is no guarantee of success, as you are still at the mercy of the various Java Runtime Environments. What's a developer to do?

The first step is to recognize when a reported "news" item is a manufactured crisis or just plain irrelevant. For example, just before this year's JavaOne event, newspapers were filled with stories of Hewlett-Packard's announcement of an "embedded VM." No report was complete without a pundit's opinion stating that this was the end of Java as we know it. Soon, everyone would be creating his or her own version of Java, and in the resulting chaos any hope of cross-platform compatibility would be doomed.

While HP's announcement undoubtedly miffed the Sun brass, for all practical purposes the significance of the story was limited to embedded systems. Unfortunately, further news coverage of HP's VM played up the diversity angle and soft-pedaled the distinction between embedded and desktop environments. And there are substantial differences. An embedded system is, by definition, a computer that is part of a larger product. The emphasis is on creating a synergy of hardware and software that reduces recurring manufacturing costs and has features distinct from the competition's products. In other words, cheap and different.

Cross-platform portability *and* increased performance. Such a deal!

Unfortunately, Sun's approach to the embedded market is neither cheap nor product-specific. The per-CPU pricing model virtually guarantees large companies will develop their own runtime environments in response. (C and C++ are the reigning languages in embedded applications, and those compiler vendors don't charge per-CPU royalties.) The migration of HP's embedded VM to the desktop and server markets is extremely unlikely for any number of reasons, a fact often glossed over, or not even mentioned, in the trade press.

At roughly the same time Hewlett-Packard was stealing the show with their JavaOne announcement, another story, more significant for developers of corporate ap-

plications, was developing. The news that Symantec's JIT technology had been adopted for both Sun's Java Development Kit and Apple's Mac OS Runtime for Java was largely ignored. Combine this with the licensing of Symantec's JIT compiler by companies such as Netscape and Novell, and fragmentation no longer seems inevitable. Indeed, it's now likely that the VM you used to develop Java applets and applications will be the same VM used in the target environment. Yes, that's right, at the same time you were seeing stories on the Balkanization of the Java market, there was strong evidence of consolidation. No lawsuits were involved, however, so the news came and went with minimal coverage.

Controlling Your Destiny

Although it's nice to read about compatibility in the marketplace, it's even nicer when you can enforce that compatibility. What can you do about it? You can control your environment, your language usage, and your assumptions. The cleanest and most obvious approach is to control your run-time environment.

Sun's approach to a control mechanism is a new product, Java Plug-in. Known during beta tests as "Activator," Java Plug-in is an add-on for browsers. By writing a half-dozen lines of HTML code, a Web page can detect the current version of a client's JVM and automate downloading a new JVM if necessary. Java Plug-in is most often used to ensure Java applets are running properly, making browser-dependent script code unnecessary and delivering on the WORA promise for browsers.

Once again, the compatibility implications of Java development were under-reported. Java Plug-in is great for providing cross-browser functionality, but its real potential lies in enterprise applications. The same mechanism can be used by JavaBeans to test the run-time environment and request the latest version.

Bad news is what gets the ink, and it seems every week brings another story of lawsuits, feuding vendors, and virtual machine incompatibilities

Intranet applications can control the Java environment within their firewalled domains by keeping a copy of the latest JVM on the server. Guaranteeing compatibility doesn't have to come at the expense of the customer's ability to run other JVMs. Finally, the Plug-in approach should provide a relatively painless method for upgrading a group of customers to JDK 1.2 features and Sun's HotSpot JVM when they are released later this year. For more information on Java Plug-in, point your browser at http://java.sun.com/products/plugin/. (Plug-in is also included in Sun's JumpStart development package.)

Plug In, Turn On

As you can imagine, Sun's Java Plug-in hasn't received Microsoft's seal of approval. Lawsuits aside, there's the fact that Plug-in is obviously designed to castrate the JVM in Microsoft browsers. Hardly the sort of behavior that generates enthusiastic acclaim from the surgery's victim.

Microsoft perceives Plug-in as inconvenient and believes developers will rebel at the notion of updating their pages to call Plug-in. This doesn't seem to be a particularly relevant criticism. First, calling Plug-in doesn't require all that much code. (See Listing 1, taken from Sun's documentation, for an example.) Second, Sun provides a free HTML conversion program to automate most of the process. Updating your Web pages for Plug-in seems considerably less difficult than, say, updating them to use DHTML.

Once you've added Plug-in tags to your page, the rest of the operation is simple. The first time a browser loads that page, it will automatically download the Plug-in software—and, by implication, the latest implementation of Sun's JRE—to the user's system. After that, future access to a Plug-in enabled page will automatically invoke the software, without user involvement.

Two points about this process are worth reiterating. First, the author of the HTML code determines how the initial installation is handled. You can make it transparent or you can offer users different options, including not continuing with installation. Second, the author also controls where Java Plug-in looks for the JRE. You can maintain your Intranet applets without broaching the safety of your firewall.

Performance Matters

Microsoft has also questioned why developers would go to all that effort to install a JVM that's slower and less compatible than its own. Leaving aside such minor technical details as RMI and support for JavaBeans, the question of performance still needs to be addressed. Sooner or later, your boss or one of your colleagues will bring up the subject and you should be prepared.

First, you'll have to remind them that speed of execution is less important than correctness. To put it in car terms, bragging about how fast you're traveling only makes sense if you're headed in the right direction and avoid crash-

Sidebar continued from previous page.

Ghost Machine. One of the more vexing aspects of Java's large memory footprint has been an inability to shoehorn a full JRE and application into the dainty memory space of PDAs like 3Com's PalmPilot. This project is an attempt to remedy that failing. You'll find some interesting ideas here, but don't expect a finished commercial product or exhaustive technical support. Read about the project at www.cs.washington.edu/homes/mcdirmid/ghost/.

Japhar. This project comes to us courtesy of the folks at Hungry Products, also known as the "Hungry Programmers." Their 1.1 JVM is described as "a work in progress," currently tracking JDK 1.1.5. This is a clean-room freeware project, designed from the beginning to be embedded inside applications. The emphasis is on tracking the Sun specifications. They're currently attempting to link the new Java VM Debugger Interface with gdb. Details at www.hungry.com/products/japhar.

Kaffe. Another freeware JVM, this time with an emphasis on JIT technology. The project and the issues of moving Java to embedded environments were the subject of an article in the February 1998 issue of *Embedded Systems Programming*. This JVM is also distinguished by the range of embedded processors it has been ported to. Progress reports and executables are at www.kaffe.org.

Other JVM projects are undoubtedly in progress as you read this. Whether those products will be made available commercially is unknown at this time. Your best bet for up-to-date information is to periodically search the Java-related Usenet discussion groups via an archive such as www.dejanews.com or check out our discussion groups at www.java-pro.com.

—*T.S.*

There Are **Two Ways**
To Deliver An Internet Application:

Throw It Out There And See If It Sinks,

Or Debug It With ProtoSpeed.

Accelerate your development cycles and produce solid Internet applications with ProtoSpeed. ProtoSpeed's distributed application debugger enables you to avoid time consuming "trial and error" debugging during development. With ProtoSpeed, you can quickly debug and monitor client and server protocol communications, Java applets, and applications to ensure that they are bug free and ready for deployment.

ProtoSpeed accelerates development and testing cycles by enabling you to:

- Set breakpoints and filters to view and edit protocol communications between clients and servers
- Debug multiple Java applets and applications, locally or remotely, with or without source code
- Debug and edit ISAPI, NSAPI and CGI requests and responses
- Monitor interactions between any HTTP, FTP, SMTP, POP3 and IMAP4 client or server
- Debug objects running in both the Microsoft and Sun Java Virtual Machines (JVMs)
- Examine the contents of ActiveX objects, Java class files, CAB and JAR files prior to activation in a browser

Start saving time and money and deploy higher quality applications. For information on purchasing ProtoSpeed, call 1-800-477-6473, ext. 4700 or visit our web site at: http://protospeed.progress.com/wp1

PROGRESS
SOFTWARE

Progress Software, 14 Oak Park, Bedford, MA 01730 USA Tel:781-280-4000 Fax: 781-280-4901

Listing 1. Worst Case HTML for Java Plug-in. Supporting both Navigator or Internet Explorer clients requires some work, but it is hardly overwhelming.

Original APPLET tag:

```
<APPLET code="XYZApp.class" codebase="html/"
    align="baseline"
  width="200" height="200">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
  No JDK 1.1 support for APPLET!!
</APPLET>
```

Combined OBJECT and EMBED tags:

```
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-
    00805F499D93"
  width="200" height="200" align="baseline"
  codebase="http://javaweb.eng/plugin/plugin-11-
    win32.exe#Version=1,1,0,0">

<PARAM NAME="code" VALUE="XYZApp.class">
<PARAM NAME="codebase" VALUE="html/">
<PARAM NAME="type" VALUE="application/x-java-
  applet;version=1.1">
<PARAM NAME="model" VALUE="models/HyaluronicAcid.xyz">
<COMMENT>
  <EMBED type="application/x-java-applet;version=1.1"
      width="200" height="200"
    align="baseline" code="XYZApp.class" codebase="html/"
    model="models/HyaluronicAcid.xyz"
    pluginspage="http://javaweb.eng/plugin/plugin-
      install.html">
  <NOEMBED></COMMENT>
    No JDK 1.1 support for APPLET!!
  </NOEMBED>
  </EMBED>
</OBJECT>
```

ing into walls. Given the growing importance of JavaBeans in enterprise Java applications, it seems prudent to select tools that directly support their use. This point is logical, but if you're afflicted with the kibitzing of a true speed demon, logic alone will not be convincing.

The second consideration is an accurate assessment of performance. That means something more substantial than shopworn, synthetic benchmarks. Perhaps something like your application.

The third consideration is strategic compatibility. Using Java Plug-in prepares your users for a relatively unobtrusive upgrade to Sun's HotSpot technology. When it ships, you'll need to cycle back to the first and second considerations again, making sure your code still works properly, and then measuring any performance upgrade.

Going Native?

Sooner or later, you'll be offered a dance with the compatibility devil. You'll be tempted with a dramatic improvement in your application's execution speed. All you'll have to sacrifice in return is a little of that fabled Java portability by compiling your application to native code instead of Java bytecode. Relax, you're not selling your soul; you're just renting it out for a few months.

Religious arguments aside, a strong case can be made for maintaining your Java code in both source and compiled native code forms. Companies like Tower Technology and SuperCede have invested a good deal of effort in making Java reach C++ performance levels, and you might as well reap the benefits. Be forewarned, however, that native code generation is limited to just a few target platforms, and you *will* need to exercise care in maintaining your source code.

If you want a performance boost, but compilation to native code just seems too heretical, there is a way of increasing performance without sacrificing portability. For example, preEmptive Solutions makes an optimizer called "Dash-O" that provides an optimization pass after you've compiled your code. Both code size and execution times go down, yet you are still dealing with bytecode derived from your Java source. Cross-platform portability *and* increased performance. Such a deal!

The migration of HP's embedded VM to the desktop is extremely unlikely

Final Words

The problem with tackling cross-platform development is that after you've covered the bases of general advice and technology trends, what remains is an endless stream of details.

For example, it seems simple enough to advise developers not to make assumptions about how threads work on the machine beneath the JVM, but it is all too easy to get snared in subtle, but important, distinctions. Code to open a socket and listen for connection requests might look innocuous and work fine on Windows NT, only to stutter and stall on a Mac. (The Mac uses cooperative threads, and must simulate the preemptive threads of other platforms.) If you're not careful, you can generate "pure" Java source that has wildly different performance characteristics, and you'll lose valuable time because you've come to expect behavior that is characteristic of your development machine rather than what is specified in the JVM manual. On many of these issues, the only substitute for lots of personal experience is reading voraciously.

But enough of the problems! For all the naysaying and pesky details, the simple fact is when it comes to cross-platform portability, we are far better off than we were just a few years back. No company has better reason to understand the pitfalls of a complex, heterogeneous environment than IBM, the very company that has become a model of rapid Java adoption. When confronted with a pack of Java skeptics at last March's JavaOne, IBM's VP for e-business software, Patricia Sueltz put it best: "No, we don't have 'Write Once, Run Anywhere' yet. But we do have 'Write Once, Run Many Places', which is better than where we used to be." ■

Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

Redefining Portability
Locator+ Code: JP0898TS

New and Improved JClass Beans now with Databinding*

Whether your "enterprise data" is on Wall Street or Main Street, JClass JavaBeans give your GUI the professional edge. That's because with the new 3.5 release, JClass Field, JClass LiveTable and JClass Chart all bind automatically to your database making RAD development a snap. And now with JClass HiGrid, you can build advanced master-detail GUIs in just minutes.

Rich functionality for every Java project! Whether or not you need RAD databinding, or even if you develop without an IDE, you'll find JClass products help you build rock-solid, robust Java GUIs quickly and easily. JClass JavaBeans support all major JDK versions, and work with AWT or JFC/Swing. So no matter how quickly JDK releases change, you can be sure that KL Group will have the JavaBeans you need to do the job. Maybe that's why Price Waterhouse recently called KL Group *the leading vendor of Java components.*

JCLASS WORKS WITH

✓ JDK 1.0.2 (AWT)	✓ Microsoft Visual J++
✓ JDK 1.1 (AWT or JFC)	✓ Penumbra SuperMoJo
✓ JDK 1.2 (JFC)	✓ Powersoft PowerJ
✓ Bluestone Sapphire/Web	✓ SilverStream
✓ Borland JBuilder	✓ Sun Java WorkShop
✓ IBM VisualAge for Java	✓ SuperCede
✓ IST Visaj	✓ Symantec Visual Café

Upgrade to JClass 3.5 for the best in JavaBean functionality and performance. All JClass 3.5 products are available for free evaluation, so if you're using an old version, or a version that came bundled with your IDE, give JClass 3.5 a try. Free evaluations are available on the Web, or by contacting KL Group.

Save with JClass Suites
JClass products are now available in Standard and Enterprise Suites including Gold Support and Subscription at attractive prices.

KL GROUP
Software Development Productivity

KL Group Inc. Toll-Free: 1-800-663-4723 Tel: 416-594-1026 Fax: 416-594-1919
KL Group Europe B.V. Tel: +31 (0)20 679 95 03 Fax: +31 (0)20 470 03 26
www.klg.com

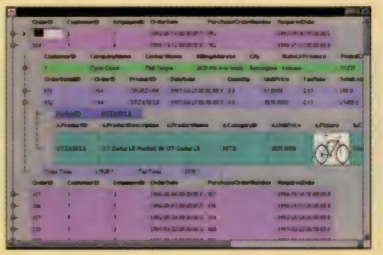
* Databinding features requires purchase of JClass DataSource, or use of Borland JBuilder 2 Professional or C/S edition, or Symantec Café 2.5 dbBE edition. JClass, JClass BWT, JClass Chart, JClass Field, JClass HiGrid and JClass LiveTable are trademarks of KL Group Inc. All other product names are trademarks or registered trademarks of their respective owners. JP002

JavaBeans!

New!

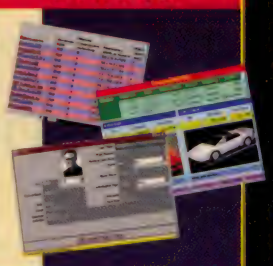
JCLASS HIGRID

An innovative RAD outline-grid, combined with our hierarchical JClass DataSource, lets you manage, display and update master-detail relation data with one easy-to-use component. All you do is specify master and detail relationships for each level of the tree-view hierarchy, and JClass HiGrid takes care of the rest!



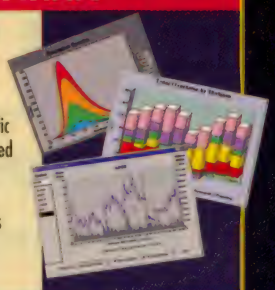
JCLASS LIVETABLE

The most advanced, high-performance and feature-rich grid/table available. Ideal for static, dynamic or db-bound data. The ideal upgrade from JTable!



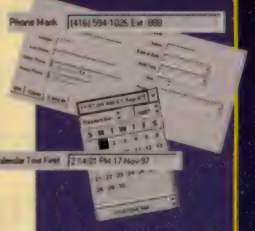
JCLASS CHART

The ultimate in Java charting and graphing, with sophisticated business and scientific graph types. Designed for superior performance with flexible data sources and interactive features.



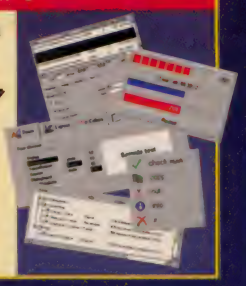
JCLASS FIELD

Delivers time-saving data input and validation for a wide range of data types. Includes masked-fields, dates (calendars), times and currencies.



JCLASS BWT

The essential companion collection for AWT users. Includes Tab Manager, Multi-Column List, Tree/Outliner plus many high-functionality, drop-in replacements for AWT controls.



EVALUATE/ORDER ONLINE
www.klg.com

Connecting to JDBC

JDBC helps Java deliver on the "write once, run anywhere" promise in database programming

by George Reese

Database programming has traditionally been a technological Tower of Babel. Programmers are faced with dozens of database products, and each one talks to our applications in its own private language. If your application needs to talk to a new database engine, you have to teach it (and yourself) a new language. As Java programmers we shouldn't have to worry about such translation issues. Java promises us the ability to "write once, compile once, and run anywhere," so it should deliver this promise with database programming as well.

> *These common thoughts are expressed in a shared public language, consisting of shared signs...a sign has a "sense" that fixes the reference and is "grasped by everybody" who knows the language...*
> —Noam Chomsky
> *Language and Thought*

The Java Database Connectivity (JDBC) API gives us a shared language through which our applications can talk to database engines. Following in the tradition of its other multiplatform APIs, such as the AWT, JDBC provides us with a set of interfaces that create a common point for database applications and database engines to meet. In this article, we'll discuss the basic interfaces that JDBC provides.

What Is JDBC?

Working with leaders in the database field, JavaSoft developed a single API for database access—JDBC—with three main goals in mind:

- JDBC should be a SQL-level API.
- JDBC should capitalize on the experience of existing database APIs.
- JDBC should be simple.

A SQL-level API means that JDBC allows us to construct SQL statements and embed them inside Java API calls. In short, you're using SQL. But JDBC lets you smoothly translate between the database and Java application worlds. Your results from the database, for instance, are returned as Java variables, and access problems get thrown as exceptions.

Because of the confusion caused by the proliferation of proprietary database access APIs, the idea of a universal database access API to solve this problem is not a new one. JavaSoft drew upon the successful aspects of one such API, Open DataBase Connectivity (ODBC). ODBC was developed as a single standard for database access in the Windows environment. Although the industry has accepted ODBC as the primary means of talking to databases in Windows, it doesn't translate well to Java. ODBC is a C API that requires intermediate APIs for other languages. But even for C developers, ODBC suffers from an overly complex design that has made its transition outside the Windows environment a failure. ODBC's complexity arises from the fact that intricate, uncommon tasks are wrapped in the API with its simpler and more common functionality.

In addition to ODBC, JDBC is heavily influenced by existing database programming APIs such as X/OPEN SQL Call Level Interface. JavaSoft wanted to reuse key abstractions from these APIs, which would ease acceptance by database vendors and capitalize on the existing knowledge capital of ODBC and SQL CLI developers. JavaSoft also realized that deriving an API from existing ones can provide quick development of solutions for database engines that support the old protocols. Specifically,

George Reese holds a degree in philosophy from Bates College in Maine. He developed the world's first JDBC driver (mSQL-JDBC), and runs one of the oldest LPMuds on the Internet, Nightmare LPMud. This article is adapted from *Database Programming with JDBC and Java*. Copyright 1997, by O'Reilly & Associates. Reproduced by permission. All rights reserved.

JavaSoft worked in parallel with Intersolv to create an ODBC bridge that maps JDBC calls to ODBC calls, giving Java applications access to any database management system (DBMS) that supports ODBC.

JDBC attempts to remain as simple as possible while providing developers with maximum flexibility. A key criterion employed by JavaSoft is asking whether database access applications read well. The simple and common tasks use simple interfaces, in fact, three interfaces handle a vast majority of database access. JDBC also provides several other interfaces for handling more complex and unusual tasks.

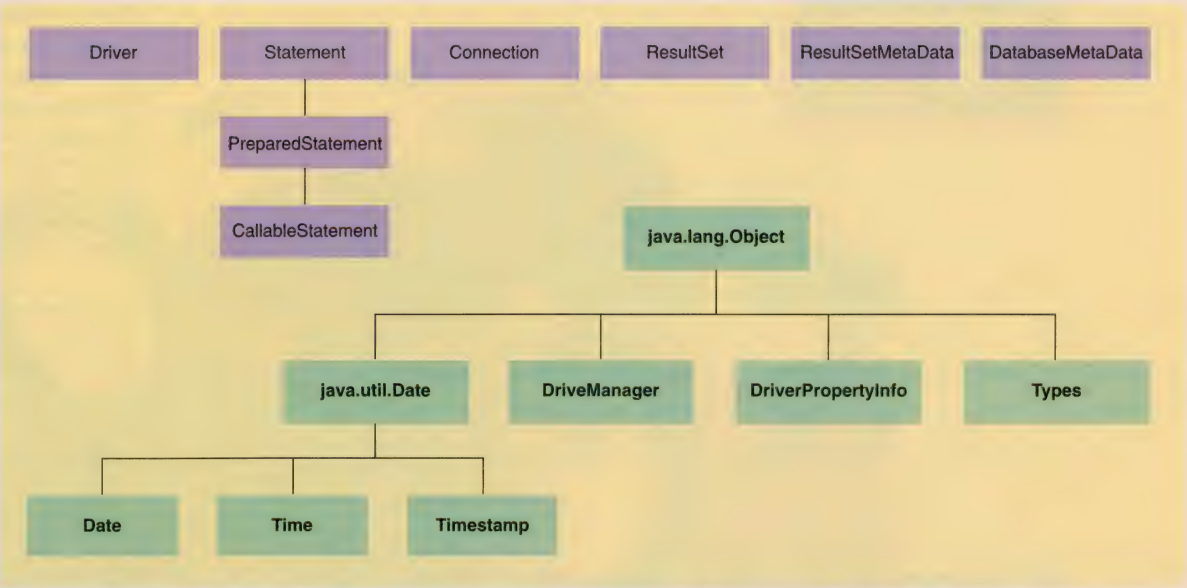


Figure 1. The Structure of JDBC. The classes and interfaces of java.sql, the JDBC API package.

The Structure of JDBC. JDBC accomplishes its goals through a set of Java interfaces, each implemented differently by individual vendors. The classes that implement the JDBC interfaces for a particular database engine are called a JDBC "driver." When building a database application, you don't have to think about the implementation of these underlying classes; the whole point of JDBC is to hide the specifics of each database and let you worry about your application. Figure 1 shows the JDBC classes and interfaces.

A database query for any database engine requires you to connect to the database, issue your SELECT statement, and process the result set. Listing 1 shows the code for a simple SELECT application from the Imaginary JDBC Driver for Mini-SQL. (Despite its name, this is a real driver, see http://Hughes.com.au for more information.) The application is a single class that gets all of the rows from a table in an mSQL database on my Sun box. First, it connects to the database using a database connection under my user id, `borg`, from the JDBC `DriverManager` class. It uses that database connection to create a Statement object, which performs the SELECT query. A ResultSet object then provides the application with the `key` and `val` fields from the `t_test` table.

Listing 1. A Simple SELECT Application. This application is a single class that gets all of the rows from a table in an mSQL database.

```
import java.sql.*;

public class SelectApp {
  public static void main(String args[]) {
    String url = "jdbc:msql://athens.imaginary.com:4333/
      db_web";

    try {
      Class.forName("imaginary.sql.iMsqlDriver");
    }
    catch( Exception e ) {
      System.out.println("Failed to load mSQL driver.");
      return;
    }
    try {
      Connection con = DriverManager.getConnection(
        url, "borg", "");
      Statement select = con.createStatement();
      ResultSet result = select.executeQuery(
        "SELECT key, val FROM t_test");

      System.out.println("Got results:");
      // process results one row at a time
      while(result.next()) {
        int key = result.getInt(1);
        String val = result.getString(2);

        System.out.println("key = " + key);
        System.out.println("val = " + val);
      }
      select.close();
      con.close();
    }
    catch( Exception e ) {
      e.printStackTrace();
    }
  }
}
```

Although the industry has accepted ODBC for talking to databases in Windows, it doesn't translate well to Java

If you have Java experience, you should be able to understand the flow of the code in Listing 1 without any knowledge of JDBC. There are no references to specific database engine classes. Instead, the code uses JDBC interfaces to provide a facade for the DBMS-specific implementation. The JDBC implementation, in turn, performs the database access behind the scenes.

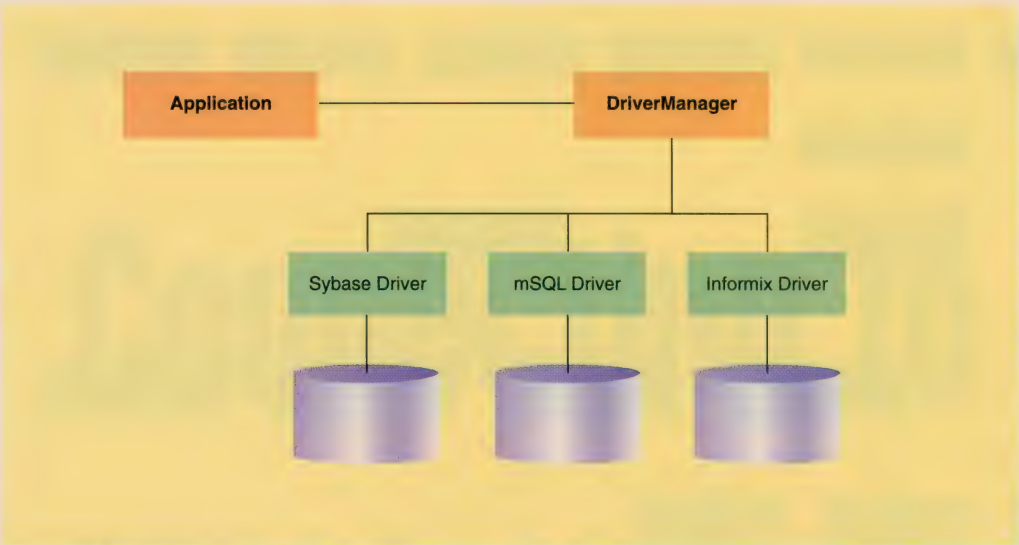


Figure 2. JDBC Implementation. JDBC shields an application from the specifics of individual database implementations.

In this simple application, the SelectApp class asks the JDBC DriverManager to hand it the proper database implementation based on a database URL. The database URL looks similar to other Internet URLs. The actual content of the URL is loosely specified as `jdbc:subprotocol:subname`. The `subprotocol` identifies which driver to use, and the `subname` provides that driver with any required connection information. In testing the Imaginary JDBC Implementation for mSQL, the URL is http://jdbc:msql:/athens.imaginary.com:4333/db_web. Translated, the URL says to use the mSQL JDBC driver to connect to the database db_web on the server running at port 4333 on athens.imaginary.com. Each URL is specific to the JDBC implementation being sought. Whatever its format, the primary function of a database URL is to uniquely identify the implementation needed by the application and pass that implementation any information it needs in order to connect to the proper database.

Databases and Drivers. Once your database engine is installed and your database is set up, you'll need a JDBC driver for the database engine. You can find an mSQL JDBC driver at www.imaginary.com/Java. Commercial database engines, like Oracle, have commercial JDBC drivers. Most offer a free trial period for experimenting with the driver. Follow the install instructions for the driver you choose, and remember that some JDBC drivers require to you install native code on client machines. To help you understand what different drivers require, JavaSoft has defined the following driver categorization system:

Type 1—These drivers use a bridging technology to access a database. The JDBC-ODBC bridge that comes with the JDK 1.1 is a good example. It provides a gateway to the ODBC API. Implementations of that API do the actual database access. Bridge solutions generally require software to be installed on client systems.

Type 2—Type 2 drivers are native API drivers. This means that the driver contains Java code that calls native C or C++ methods provided by the individual database vendors that perform the database access. Again, this solution requires software on the client system.

Without JDBC, only disparate, proprietary database access solutions exist

Type 3—Type 3 drivers provide a client with a generic network API that is translated into database specific access at the server level. In other words, the JDBC driver on the client uses sockets to call a middleware application on the server that translates the client requests into an API specific to the desired driver. This type of driver is extremely flexible because it doesn't require code to be installed on the client and a single driver can provide access to multiple databases.

Type 4—Using network protocols built into the database engine, type 4 drivers talk directly to the database using Java sockets. This is the most direct and pure Java solution. In nearly every case, this type of driver will come only from the database vendor. Table 1 lists the driver vendors that were public at the time publication. See http://splash.javasoft.com/jdbc/jdbc.vendors.html for a current list of JDBC vendors.

Alternatives to JDBC. Without JDBC, only disparate, proprietary database access solutions exist. These proprietary solutions force developers to build a layer of abstraction on top of them in order to create database independent code. Only after that abstraction layer is complete can you write the application. In addition, your experience with that abstraction layer does not translate immediately to other projects or other employers, who are almost certainly using their own abstraction layers to provide access to a variety of database engines.

Of course, the ODBC specification exists to provide this universal abstraction layer for languages like C and C++, as well as popular development tools such as Delphi, PowerBuilder, and Visual Basic. Unfortunately, ODBC does not enjoy the platform independence of Java. Using the JDBC interface design, however, your server application can pick the database at run time based on which client is connecting. The ATM front-end of the banking application, for example, could have legacy data stored in an Oracle system that is not yet converted to the Informix system against which the rest of the application is running. You could write the application without writing special access code for the legacy data. The ATM client passes the Oracle URL when requesting the connection. Once its data has been converted to Informix, all you have to do to switch the application to the new database is tell the ATM machine about the new database URL. No new code needs to be written for the migration.

Connecting to the Database

Any JDBC application we write needs to be able to run from start to finish without referencing a specific JDBC implementation. Figure 2 shows how an application uses JDBC to talk to one or more databases without knowing any of the details concerning the driver implementation for that database. An application uses JDBC as an interface through which it passes all of its databases requests.

When you write a Java database applet or application, the only driver-specific information JDBC requires is the database URL. You can build your application so that it derives the URL at run time based on user input or applet parameters. Using the database URL, a user

GRAB IT WHILE IT'S HOT
OFFER EXPIRES SOON!!

LIMITED TIME
INTRODUCTORY
OFFER $995

IS THIS YOUR JAVA TOOL'S IDEA OF SUPPORTING TEAM DEVELOPMENT?

Then you obviously don't use PARTS® for JAVA™ Professional, a tool set designed to fully support teams building distributed applications.

PARTS was created by developers with over 10 years experience designing object-oriented development tools for team programming. Our experience will make a difference in your team's productivity.

PARTS will coordinate your team's efforts with integrated support for PVCS, making it simple to track changes, compare versions, and manage your development projects. Plus, our unique tools help you understand how your code interacts with other pieces of the system, enabling you to clearly see the impact of changes.

PARTS has everything you need to be productive:

- Visual programming—just drag, drop and deploy
- JDBC support for fast, easy database applications
- JFC/Swing for powerful interfaces
- CORBA for distributed applications
- JavaBeans for quick application assembly
- OracleLite, OrbixWeb 3.0, PVCS, jKit/Grid— All integrated for a combined value of $1700!

And all backed by our world class training and consulting program chosen by both Oracle® and Sun®.

Call now for more information on how PARTS will make your team more productive—
1.800.973.4777

ObjectShare®

16811 Hale Ave., Suite A, Irvine, CA 92606 www.objectshare.com

PARTS is a registered trademark of ObjectShare, Inc. All other referenced products are trademarks and/or registered trademarks of their respective manufacturers.

PARTS® FOR JAVA™
PROFESSIONAL

BE PRODUCTIVE.

name, and password, your application first requests a java.sql.Connection implementation from the DriverManager. The DriverManager in turn searches through all of the known java.sql.Driver implementations for one that will connect with the URL you provided. If it exhausts all of the implementations without finding a match, it throws an exception back to your application.

Once a Driver recognizes your URL, it creates a database connection using the user name and password you specified. It then provides the DriverManager with a java.sql.Connection implementation representing that database connection. The DriverManager then passes that Connection object back to the application. In your code, the entire database connection process is handled by this one-liner:

```
Connection con =
  DriverManager.getConnection(url, uid,
  password);
```

You're probably wondering how the JDBC DriverManager learns about a new driver implementation. The DriverManager keeps a list of classes that implement the java.sql.Driver interface. Somehow, somewhere, your application needs to load the Driver implementations for any potential database drivers. JDBC requires a Driver class to register itself with the DriverManager when it gets loaded. Loading a Driver class enters it in the DriverManager's list. Your only task is to determine the somehow and somewhere of loading those Driver implementations. You have several alternatives.

You can explicitly call `new()` to load your driver's implementation of Driver. In other words, you hard code the loading of a Driver implementation in your application. This alternative is the least desirable because it requires a rewrite and recompile if your database or database driver changes.

You can use the `jdbc.drivers` property. The DriverManager will load all classes listed in this property automatically. This alternative works well for applications with a command-line interface, but might not be so useful in GUI applications and applets because you can specify properties at the command line or in environment variables. While environment variables do work for GUI applications, you cannot rely on them in Java applets.

Finally, you can load the class using `Class.forName("DriverImplementationClass");` This Java mechanism loads classes dynamically based on a String with the class name.

Listing 2. A Simple Database Connection. This puts the process of connecting to the database into a more concrete format.

```
package example.sql;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

/**
   * The SimpleConnection class is a command line
   * application that accepts
   * the following command line:
   * java SimpleConnection DRIVER URL UID PASSWORD
   * If the URL fits the specified driver,
   * it will then load the driver and
   * get a connection.
   */
public class SimpleConnection {
   static public void main(String args[]) {
      Connection connection = null;

      // Process the command line
      if( args.length != 4 ) {
         System.out.println("Syntax: java
            SimpleConnection " +
            "DRIVER URL UID PASSWORD");
         return;
      }
      try { // load the driver
         Class.forName(args[0]);
      }
      catch( Exception e ) { // problem loading driver,
         // class not exist?
         e.printStackTrace();
         return;
      }
      try {
         connection = DriverManager.getConnection(
            args[1]. args[2], args[3]);
         System.out.println("Connection successful!");
         // Do whatever queries or updates you want here!!!
      }
      catch( SQLException e ) {
         e.printStackTrace();
      }
      finally {
           try {
                connection.close();
           }
           catch( SQLException e ) {
              e.printStackTrace();
           }
      }
   }
```

Listing 3. UpdateApp. This class performs an INSERT at a particular URL, then reports back the number of rows inserted.

```
import java.sql.*;

class UpdateApp {
   public static void main(String args[]) {
      Connection connection = null;

      if( args.length != 2 ) {
         System.out.println(
            "Syntax: <java UpdateApp [number] [string]>");
         return;
      }
      try {
         Class.forName("imaginary.sql.iMsqlDriver");
         String url = "jdbc:msql://athens.imaginary.com:
            4333/Testdb";
         con = DriverManager.getConnection(url, "borg", "");
         Statement s = con.createStatement();
         String test_id = args[0];
         String test_val = args[1];
         int update_count =
            s.executeUpdate("INSERT INTO t_test (
               test_id, test_val) " +
               "VALUES(" + test_id + ", '" + test_val + "')");

         System.out.println(update_count + " rows inserted.");
         s.close();
      }
      catch( Exception e ) {
         e.printStackTrace();
      }
      finally {
         try { con.close(); }
         catch( SQLException e ) { e.printStackTrace(); }
      }
   }
}
```

Listing 4. Fear of Commitment. After ensuring that autocommit is off, this application either commits two updates together or rolls them both back.

```
import java.sql.*;

class UpdateLogic {
  public static void main(String args[]) {
    Connection connection = null;

    if( args.length != 2 ) {
      System.out.println(
        "Syntax: <java UpdateLogic [number] [string]>");
      return;
    }
    try {
      Class.forName("imaginary.sql.iMsqlDriver");
      String url = "jdbc:msql://athens.imaginary.com:
        4333/db_test";
      Statement s;

      con = DriverManager.getConnection(url, "borg", "");
      con.setAutoCommit(false);
        // make sure auto commit is off!
      s = con.createStatement();
        // create the first statement
      s.executeUpdate("INSERT INTO t_test (
        test_id, test_val) " +
          "VALUES(" + args[0] + ", '" +
            args[1] + "')");
      s.close();
        // close the first statement
      s = con.createStatement();
        // create the second statement
      s.executeUpdate("INSERT into t_test_desc (
        test_id, test_desc) " +
            "VALUES(" + args[0] +
            ", `This describes the test.')");
      con.commit();
        // commit the two statements
      System.out.println("Insert succeeded.");
      s.close();// close the second statement
    }
    catch( SQLException e ) {
      if( con != null ) {
        try { con.rollback(); }// rollback on error
        catch( SQLException e ) { }
      }
      e.printStackTrace();
    }
    finally {
      try { con.close(); }
      catch( SQLException e ) { e.printStackTrace(); }
    }
  }
}
```

This static method in the Class class returns the Java class represented by the String you pass. A side effect of this behavior is that the static constructor in that class will be executed. JDBC requires that the static constructor for a Driver implementation load an instance of the Driver. As I said earlier, when you load a Driver, it registers itself with the DriverManager.

I use the third alternative almost exclusively since it requires no hard-coded class names and runs well in all Java environments.

The JDBC Classes for Creating a Connection. As you can see from the process flow above, JDBC uses one class (java.sql.DriverManager) and two interfaces (java.sql.Driver and java.sql.Connection) to connect to a database.

java.sql.Driver

Unless you're writing a custom JDBC implementation, you should never have to deal with this class from your application. It simply gives JDBC a launching point for database connectivity by responding to DriverManager connection requests and providing information about the implementation in question.

java.sql.DriverManager

Unlike most other parts of JDBC, DriverManager is a class instead of an interface. Its main responsibility is to maintain a list of Driver implementations and present an application with one that matches a requested URL. The DriverManager provides `registerDriver()` and `deregisterDriver()` methods, which allow a Driver implementation to register itself with the DriverManager or remove itself from that list. You can get a list of registered drivers through the `getDrivers()` method.

java.sql.Connection

The Connection class represents a single logical database transaction. Use the Connection class to send a series of SQL statements to the database. This class also manages the committing or aborting of those statements.

Making a database call is nothing more than creating a Statement and passing it to SQL via one of its execute methods

Listing 2 puts the process of connecting to the database into a more concrete format. When connecting to the database, this example catches a `SQLException`. This is a catchall exception for database errors. Almost any time something goes wrong between JDBC and the database, JDBC throws a `SQLException`. In addition to the information you commonly find in Java exceptions, `SQLException` provides database specific error information, such as the `SQLState` value and vendor error code. JDBC `SQLExceptions` may be chained together so you can find out everything that went wrong with a catastrophic database call.

Basic Database Access

Now that you're connected to the database, you can begin making updates and queries. The most basic database access involves writing JDBC code where you know ahead of time whether the statements you are sending are updates (INSERT, UPDATE, or DELETE) or queries (SELECT).

Basic database access starts with the Connection object you created in the previous section. When the Connection object is initially created, it's simply a direct link to the database. Use it to generate implementations of `java.sql.Statement` tied to the same database transaction. After you have used one or more Statement objects generated by your Connection, you can use it to commit or rollback the Statement objects associated with that Connection.

A Statement is exactly what its name implies—a SQL statement. Once you get a Statement object from a Connection, you have what amounts to a blank check that you can write against the transaction represented by that Connection. You don't actually assign a SQL statement to the Statement until you are ready to send the SQL to the database.

At this point it is important to know what type of SQL statement you are sending to the database, because JDBC uses a different method for sending queries than it does for sending updates. The

key difference is the fact that the method for queries returns an instance of `java.sql.ResultSet`, while the method for non-queries returns an integer. A ResultSet provides you with access to the data retrieved by a query.

Basic JDBC Database Access Classes

java.sql.Statement The Statement class is the most basic of three JDBC classes representing SQL statements. It performs all of the basic SQL statements we have discussed. In general, a simple database transaction uses only one of the three statement execution methods in the Statement class. The first such method, `executeQuery()`, takes a SQL String as an argument and returns a ResultSet object. This method should be used for any SQL calls that expect to return data from the database. Update statements, on the other hand, are executed using the `executeUpdate()` method. This method returns the number of affected rows.

Finally, the Statement class provides an `execute()` method for situations in which you don't know whether the SQL being executed is a query or update. This usually happens when the application is executing dynamically created SQL statements. If the statement returns a row from the database, the method returns true. Otherwise it returns false. The application can then use the `getResultSet()` method to get the returned row.

java.sql.ResultSet A ResultSet is a row of data returned by a database query. The class provides a series of methods for retrieving columns from the results of a database query. The methods for getting a column take the form:

```
type get type(int | String)
```

The argument represents either the column number or column name desired. A nice side effect of this design is that you can store values in the database as one type and retrieve them as another. For example, if you need a Date from the database as a String, you can get it as a String by calling `result_set.getString(1)` instead of calling `result_set.getDate(1)`.

Because the ResultSet class handles only a single row from the database at a time, the class provides the `next()` method for making it reference the next row of a result set. If `next()` returns true, you have another row to process and any subsequent calls you make to the ResultSet object will be in reference to that next row. If there are no rows left, it returns false.

Note that ResultSet objects allow only one-way navigation through rows from a query—there is no `previous()` counterpart to the `next()` method. JavaSoft is considering the addition of a RowSet class for Java 1.2 that will allow you to scroll through result sets in a manner similar to database cursors.

Clean Up. In the examples provided so far, you might have noticed that many of the objects are being closed through a `close()` method. The Connection, Statement, and ResultSet classes all have `close()`. A given JDBC implementation might require you to close these objects before reusing because they are likely holding precious database resources. It's a good idea to close any instance of these objects when you're done with them. If you do manage to close a Connection before committing with auto-commit off, any uncommitted transactions will be lost.

Modifying the Database. In Listing 1 we used JDBC to perform a simple SELECT query. Of course, we can't retrieve data from the database before it is there. Listing 3 shows the simple UpdateApp class supplied with the Imaginary JDBC Implementation for mSQL.

Again, making a database call is nothing more than creating a Statement and passing it to SQL via one of its execute methods. Unlike `executeQuery()`, however, `executeUpdate()` does not return a ResultSet (you should not be expecting any results). Instead, it returns the number of rows affected by the UPDATE, INSERT, or DELETE.

By default, JDBC commits each SQL statement as it is sent to the database; this is called *autocommit*. However, for more robust error handling, you can set up a Connection object so it issues a series of changes that have no effect on the database until you expressly send a commit. Each Connection is separate, and a commit on one has no effect on the other statements. The Connection class provides the `setAutoCommit()` method so you can turn autocommit off. Listing 4 shows a simple application that turns autocommit off and commits two statements together or not at all.

The JDBC Support Classes

JDBC provides a handful of other classes and interfaces that support JDBC's core functionality. Many of them are more SQL-friendly extensions of java.util classes like java.sql.Date and java.sql.Numeric. Others are exception classes that get thrown by JDBC calls.

Vendor	Type	Supported Databases
Agave Software Design	3	Oracle, Sybase, Informix, ODBC supported databases
Asgard Software	3	Unisys A series DMSII database
Borland	4	InterBase 4.0
Caribou Lake Software	3	CA-Ingres
Center for Imaginary Environments	4	mSQL
Connect Software	4	Sybase, MS SQL Server
DataRamp	3	ODBC supported databases
IBM	2/3	IBM DB2 Version 2
IDS Software	3	Oracle, Sybase, MS SQL Server, MS Access, Informix, Watcom, ODBC supported databases
InterSoft	3	Essentia
Intersolv	2	Oracle, Sybase
JavaSoft	1	ODBC supported databases
OpenLink	3	CA-Ingres, Postgres95, Progress, Unify
SAS	4	SAS, and via SAS/ACCESS, Oracle, Informix, Ingres, and ADABAS
SCO	3	Informix, Oracle, Ingres, Sybase, Interbase
StormCloud Development	3	ODBC supported databases
Sybase	3/4	Sybase SQL Server, SQL Anywhere, Sybase IQ, Replication Server
Symantec	3	Oracle, Sybase, MS SQL Server, MS Access, Watcom, ODBC supported databases
Visigenic	3	ODBC supported databases
WebLogic	2/3	Oracle, Sybase, MS SQL Server/ODBC supported databases

Table 1. A List of JDBC Driver Vendors. Each vendor provides drivers based on proprietary protocols.

java.sql.Types. The Types class provides constants that identify SQL data types. Each constant that represents a SQL data type that is mapped to an integer is defined by the XOPEN SQL specification.

java.sql.SQLException. The SQLException class extends the general java.lang.Exception class that provides extra information about a database error. The information provided by a SQLException includes:

- The SQLState string describing the error according to the XOPEN SQLState conventions. The different values of this string are defined in the XOPEN SQL specification.
- The database-specific vendor error code. This code is usually a number that you have to look up in the obscure reference section of your database's documentation. Fortunately, the error should be sufficiently described through the Java Exception class's `getMessage()` method.
- A chain of exceptions leading up to this one. This is one of the niftier features of this class. Specifically, if you get several errors during the execution of a transaction, you can chain them all together in this class. This is frequently useful when you have exceptions that you want to let the user to know about, but you do not want to stop processing:

```
Connection connection =
  DriverManager.getConnection(url, uid,
            pass);
catch( SQLException e ) {
 e.printStackTrace();
 while( (e = e.getNextException()) != null ) {
     // while more exceptions
   e.printStackTrace();
```

java.sql.SQLWarning and java.sql.DataTruncation. Depending on the driver you are using, non-fatal errors might occur, but they shouldn't halt application processing. JDBC provides an extension to the SQLException class called SQLWarning. When a JDBC object, like a ResultSet, encounters a warning situation internally, it creates a SQLWarning object and adds it to a list of warnings that it keeps. At any point, you can get the warnings for any JDBC object by repeatedly calling the `getWarnings()` method until it returns null.

The DataTruncation class is a special warning that a JDBC implementation throws when JDBC unexpectedly truncates a data value. A DataTruncation object is chained as a warning on a read operation and thrown as an exception on a write.

java.sql.Date, java.sql.Time, and java.sql.Timestamp. Portable date handling among database engines can be very complex. Each relational database management system (RDBMS) seems to have its own unique way of representing date information. These three classes extend the functionality of other Java objects to provide a portable representation of their SQL counterparts. The Date and Time classes represent different levels of granularity as well as different means of expressing information already found in the java.util.Date class. The java.sql.Date class, for example, provides methods to express just the date, month, and year, while the Time class works in terms of hours, minutes, and seconds. And finally the Timestamp class takes the java.util.Date class down to nanosecond granularity.

java.sql.DriverPropertyInfo. I can almost guarantee that you will never use this class. It is designed for Rapid Application Development (RAD) tools like Symantec Visual Café and Borland JBuilder. In order to provide a graphical user interface for rapid prototyping, these tools need to know what properties are required by a given JDBC implementation in order to connect to the database. Most drivers, for example, need to know the user name and password of the user connecting to the database. That and anything else the driver needs in order to connect to the database will be returned as an array of DriverPropertyInfo objects from the `java.sql.Driver getPropertyInfo()` method. Development tools can call this method to find out what information they should prompt the user for before connecting to the database.

Going Further

After this overview, you'll probably want to start exploring JDBC directly. Getting hands-on experience with JDBC should be not a problem. Not only does the JDK include a JDBC implementation, but most of the commercial development environments provide additional JDBC support.

At press time, however, Sun had just released the final specification for the JDBC 2.0 API. The new version includes support for SQL3 data types, scrollable cursors, and the storing and retrieving of Java data types. For more information, or to download the latest implementation, aim your browser at http://java.sun.com/products/jdbc/index.html. ■

Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

Connecting to JDBC
Locator+ Code: JP0898GR

FRESH HOT IMAGING

Now serving
JAVA Classes
&
JAVA Beans

ImageGear Java

Developing in JAVA? Wishing to add imaging capabilities to your application? Fast? Stop dreaming! **We are currently serving a pure JAVA imaging toolkit.** Based on our award-winning ImageGear™ technology, ImageGear JAVA™ is the hottest pick for all of you who need the extra flexibility and performance of JAVA in your development work. ImageGear JAVA™ comes in two high-octane flavors: **JAVA Beans** and **JAVA Classes**, both backed up with the AccuSoft's Image Guarantee™. Their hot features include:

- ✓ Decode popular imaging formats: TIFF, GIF, JPEG, BMP, MO:DCA, IOCA and CALS
- ✓ Image Loading and Saving
- ✓ MultiThreading
- ✓ Web-based Architecture
- ✓ Small Foot Prints
- ✓ Format Conversion
- ✓ Printing
- ✓ High Speed Display
- ✓ Image Processing

ImageGear JAVA can be yours today!
Call 800-741-7769

AccuSoft
High Performance Imaging

2 Westborough Business Park • Westborough, MA 01581 • Tel (508) 898-2770 • Fax (508) 898-9662
www.accusoft.com

© AccuSoft Corporation. All Rights Reserved. AccuSoft and ImageGear are registered trademarks of AccuSoft Corporation. High Performance Imaging, ImageGear98 are trademarks of AccuSoft Corporation. All company and brand names are trademarks or registered trademarks of their owners.

Java 1.0; Oracle Web Application Server; Oracle8

Plugging into Oracle

Putting Oracle to work in your Java applications

by Dennis Harvey and Steve Beitler

Java developers who are interested in writing applications for the Web can now add a powerful weapon to their arsenals—the Oracle Web Application Server 3.0 (OAS). After months of getting to know OAS, we're convinced that it's a lot more than a Web server. It's a high-powered application server that lets developers use the power of the Java cartridge within OAS to write Web apps in server-side Java code.

Our plan in this article is to describe the basics of OAS, with a special focus on the Java cartridge. We'll look at what makes OAS more than a conventional Web server, and give you a better idea of what OAS can do for you.

The first step in appreciating OAS is to understand Oracle's architectural framework. The Network Computing Architecture (NCA), is a development environment that is the "umbrella" architecture for all of Oracle's enterprise products. NCA is designed to enable developers to build distributed, multitiered apps from modular components, called cartridges. These cartridges can exist on any of the three tiers: client, application server, and universal data server. The intercartridge exchange provides the software bus that allows cartridges to communicate across the tiers. This bus, which delivers some of the real power of NCA, is built on top of CORBA.

Oracle has a number of products that make up NCA. Oracle8, the most recent version of Oracle's flagship RDBMS, is the database server, or the universal data server. This label reflects the fact that Oracle8 incorporates object relational data types. Objects allow the database to be extended by using "data cartridges." These data cartridges are made up of a number of object data types that include attributes. The attributes define the state of an object as well as methods that delineate the operations that are possible on an object. These characteristics enable developers to extend the power of the universal data server to address new problem domains.

Also included in NCA are OAS, Designer/2000, and Developer/2000. Designer/2000 is a modeling tool that can automatically create Web apps based on modeling data in a repository. Developer/2000 is a visual development environment that can be used with PL/SQL to create client/server apps. As a point of comparison, Oracle's NCA addresses roughly the same problem space as Microsoft's Distributed interNet Applications (DNA) architecture.

OAS is the middle tier of Oracle's NCA and the focus of this article. Let's take a closer look.

Oracle Web Application Server

Oracle seems to have two key goals for OAS. The first is to deliver an application server platform that is as powerful and flexible as conventional client/server platforms. The second is to enable the universal access to applications that thinner-client, multitiered application architectures make possible. Distributed object technology is the key to achieving these goals. The distributed nature of the architecture is what makes OAS so significant for Java developers.

Let's look at this difference more closely. Conventional Web servers listen for HTTP requests and respond by sending an HTTP response and the associated data to the browser. Thus, HTTP is best suited to static Web content, such as HTML pages and graphics.

The Common Gateway Interface (CGI) spec is a step toward enabling Web servers to become true application servers. However, CGI's great strength—its simplicity—is also its weakness. The CGI environment lacks the application services that are essential to most corporate apps, such as centralized application administration, session and state management, transactions, and logging. In addition, the

This article is based on *The Developer's Guide to Oracle Web Application Server 3*, by Dennis Harvey and Steve Beitler, Addison Wesley Longman Inc. 1998. This material is reproduced with permission of Addison Wesley Longman Inc. Copyright © 1998. The author can be reached harvey@newpath.com.

CGI architecture scales poorly, because a new CGI process image is spawned to handle each request.

Vendors have responded with proprietary server-side "plug-in" APIs such as Microsoft ISAPI and Netscape NSAPI. Your application takes the form of a shared library (or DLL on Windows NT) that runs in the same process space as the HTTP server. This architecture offers much better performance than conventional CGIs. This is still a partial solution, however, because it does not provide a complete suite of application services. Scaling your application with this architecture requires that you replicate your Web servers and the associated shared libraries.

But the architecture of OAS is more advanced, flexible, and complicated. Within that environment there are four key entities (see to Figure 1): HTTP listeners, dispatchers, the Web Request Broker (WRB), and cartridges. Under OAS, these components and services are decoupled. They exist as distributed object services that can communicate over a network. For example, the HTTP servers (listeners) are physically separated from the apps, which are implemented as cartridges.

OAS works with the major Web servers: Netscape and Microsoft IIS (Internet Information Server). The popular Apache listener will be supported in the next release—OAS 4.0. OAS also includes its own HTTP listener, Spyglass, which handles HTTP requests for static documents and standard CGI programs. Requests for OAS application cartridges are handled by a dispatcher, which determines the kind of application cartridge to route the request to.

The WRB, the real "guts" of OAS, contains the CORBA-compliant Object Request Broker (ORB). CORBA provides the architecture and related services for what are really remote procedure calls to objects. WRB is what enables all the other entities within OAS to be distributed and to interact over a network.

Finally, applications for the OAS environment are developed and deployed as cartridges, which can be either tool cartridges or application cartridges. OAS includes a number of tool cartridges: PL/SQL, Perl, LiveHTML, ODBC, VRML, and, of course, Java. Application cartridges are developed using one of the tool cartridges or the WRB C API, which is loosely analogous to Microsoft ISAPI or Netscape NSAPI.

This is a pretty quick run-through of OAS, but there are several strengths of this environment that Java developers can take advantage of, particularly its distributed, scalable nature, cartridges, and ICX.

There are also several application services within OAS worth noting. We'll now look at these briefly.

Application Services

OAS delivers a major improvement with transactions. Real database transactions have been noticeably absent from Web application development environments. We define these services as the robust transactions supported in two-tier client/server apps using commit and rollback, and in three-tiered apps using TP monitors.

The stateless nature of HTTP—which requires each HTTP request and response to take place without reference to any previous requests—makes transaction services difficult. Another complication is the absence of a universal mechanism to identify clients. OAS, by contrast, has two mechanisms for providing such services, as well as related capabilities for identifying sessions.

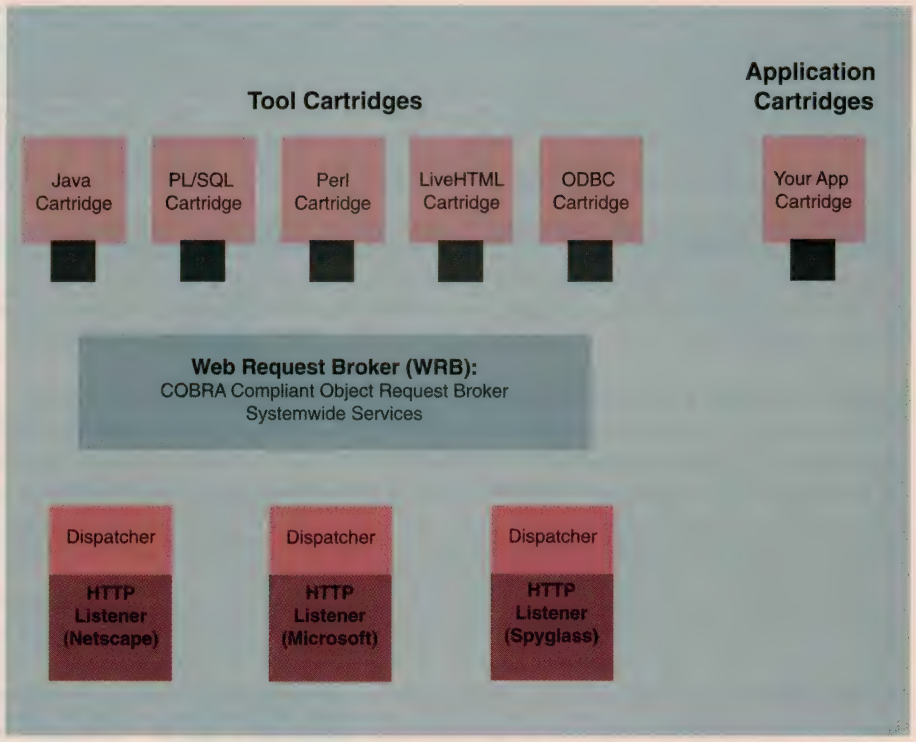


Figure 1. The Oracle Application Server Architecture. Unlike other designs you may have seen, OAS is designed as a distributed object architecture. The system components and services are decoupled and communicate over a network.

Security is a significant challenge in the distributed environment of OAS, which addresses this challenge with some notable tools and services. In OAS, security can be implemented at the listener, dispatcher, and cartridge levels. The OAS Authorization Server provides services that manage security across all the cartridges and authentication services for the dispatcher. There are two ways in which security can be implemented within a cartridge. One approach is to use calls in the WRB API to make requests of the Authentication Server; the other is for a cartridge to self-authenticate, using the username and password that the browser supplies.

The first step in appreciating OAS is to understand Oracle's architectural framework

Another noteworthy service is intercartridge exchange (ICX). ICX provides a way to build applications from HTML-generating components. For example, one piece of your Web page might be generated using the Java cartridge, and another with the PL/SQL cartridge.

OAS also provides centralized administration for listeners, services, and cartridges, as well as a centralized logging facility. The Web Application Server (WAS) Manager, included in OAS, is used to configure and manage a distributed OAS installation. The WAS Manager is also used to control the centralized logging service in OAS.

The other services within OAS are less essential to this discussion, so we won't detail them. Instead, we'll move to the heart of our discussion: the Java cartridge itself.

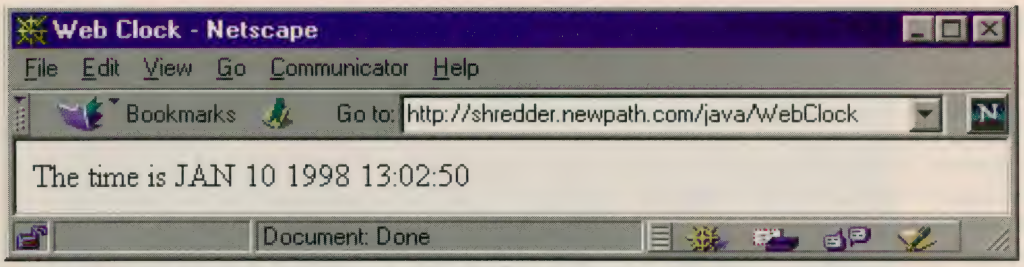

Web Clock - Netscape
File Edit View Go Communicator Help
Bookmarks Go to: http://shredder.newpath.com/java/WebClock
The time is JAN 10 1998 13:02:50
Document: Done

Figure 2. Hello World! Do You Have the Time? Using the services of the Java cartridge doesn't have to be difficult or complicated. The source code for this Web Clock application is only slightly longer than your first programming exercise.

Inside the Java Cartridge

The Java platform's strengths are well known to Web developers, and Oracle has made Java a strategic focus of its NCA. For example, in the pending releases, Oracle8 will allow developers to program database-stored procedures and triggers in Java.

This focus is also reflected in the Java cartridge within OAS. The current release of OAS is based on Version 1.02 of the Java Developers' Kit (JDK). OAS 4.0, the next release currently under development, will support JDK 1.1.x. The Java cartridge within OAS enables efficient interaction with Oracle PL/SQL procedures by using the `p12java` utility to create Java wrapper classes automatically. Beyond PL/SQL, Java Database Connectivity (JDBC) delivers a powerful, vendor-independent class library for interacting with any database that has the required driver.

The Java cartridge also includes a Web Toolkit for Java in the form of class libraries. This toolkit's container-component model allows developers to create dynamic Web pages. It also provides HTTP utility classes as well as classes that permit interaction with an Oracle database using PL/SQL.

Perhaps the quickest way to showcase the power of OAS, and the Java cartridge within it, is to implement a very simple application (see Figure 2). We call it a Web Clock; it displays the date and time. The application source code using the Java cartridge is straightforward:

```
import oracle.html.*;
import java.util.Date;

public class WebClock {
   public static void main(String
    args[])
   throws HtmlException {
       HtmlHead head = new HtmlHead("Web
        Clock");
       HtmlBody body = new HtmlBody();
       HtmlPage page = new HtmlPage(head,
        body);
       body.addItem("The time is " +
               new Date().toString());
       page.printHeader();
       page.print();
   }
}
```

Our simple Web Clock application highlights another strength of the OAS cartridge approach—it allows your to develop you applications using whatever language is appropriate. Our Java could just as easily have been PL/SQL, Perl, LiveHTML, or even C++!

Programming the Java Cartridge

Now that you've been introduced to the basic structure of the Java cartridge in OAS, it's time to discuss programming within that cartridge. The core Java Web Toolkit is comprised of three packages that can be referenced from Java code. To do this, you'll use three import statements:

```
import oracle.html.*
import oracle.plsql.*
import oracle.rdbms.*
```

The .html package contains classes that generate HTML in an object-oriented manner, and the classes within the plsql package represent data types that correspond to PL/SQL data types. The oracle.rdbms package contains a Sessions class, which connects to an Oracle RDBMS to provide access to PL/SQL packages; procedures and functions; and a ServerException class that handles PL/SQL errors. Let's look briefly at each of these packages.

The oracle.html package in classes is used to generate dynamic HTML. It's also possible to generate dynamic HTML by writing your HTML as a character string to System.out. To demonstrate the differences, we've written another sample program using both approaches (see Listings 1 and 2). Listing 1 shows how to generate a simple page without using Java Web Toolkit classes. Listing 2 shows how to generate the same page using what the Toolkit calls a component-based dynamic page construction approach. The `HelloWorldWebToolkit` class uses the HtmlHead, HtmlBody, and HtmlPage classes from the Java Web Toolkit.

There are four key classes within the oracle.html package: the Item class, the IHtmlItem interface, the Container class, and the CompoundItem class.

The Item class is an abstract base class for all other classes that generate the HTML tags that define the body of an HTML page. The Item class incorporates methods that enable developers to specify the attributes of each HTML tag. This class also includes the `print()` and `toHTML()` methods. The `print()` method is roughly equivalent to the `htp` procedures in the PL/SQL toolkit, while the `toHTML()` method is the counterpart to the PL/SQL toolkit's `htf` functions.

The IHtmlItem interface includes these two methods. This interface is used in classes that don't extend the Item superclass, such as Comment, HtmlFile, HtmlHead, HtmlPage, and MetaInfo.

The Container class is a group of objects that

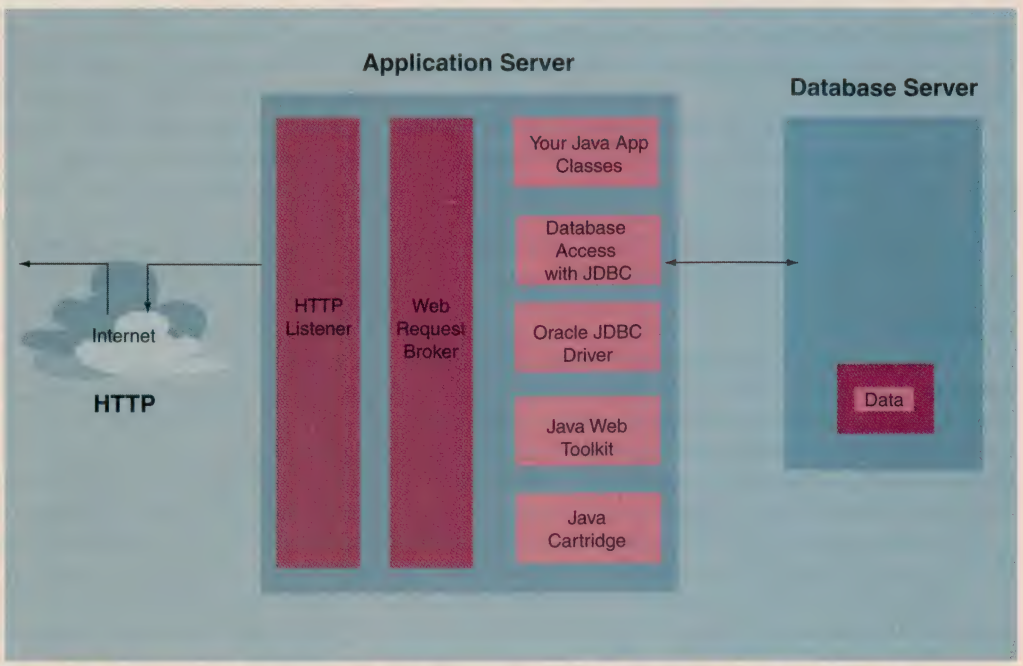


Figure 3. Distributed Logic for your Web Application. Using JDBC, more of your application logic resides on the middle tier. In a more traditional design, more of the logic would reside on the data server.

Fiercely independent. Technically advanced.

SIGS Conference for JAVA™ Development

October 18–22, 1998

Presented by SIGS CONFERENCES Java REPORT

San Jose Convention Center • San Jose, CA.

THE MEGA EVENT FOR SERIOUS JAVA USERS.

Join more than 5,000 of your peers at the **SIGS Conference for Java Development,** and see how to get the most out of utilizing Java. Plan to attend this vendor-neutral event to get techniques, practical solutions, proven strategies, and usable advice straight from the experts.

SIGS Conference for Java Development offers a unique combination of technical sessions, night school sessions and pre-and post-conference full-day tutorials. In addition, the two day exhibition on October 19–20 will feature many leading Java vendors showcasing the latest tools and services.

Whether you're exploring Java for future application development or are already on your way to developing enterprise strength applications, you will benefit from exposure to so many industry experts and vendors.

SIGS Conference for Java Development is conveniently located in the San Jose Convention Center–the heart of Silicon Valley and the hub of Java development.

Technical Chair

Class sessions were hand-picked and planned with the supervision of our Conference Technical Chair, Dwight Deugo, Editor of the JAVA REPORT.

What Makes This Event So Valuable?

- You'll learn about new Java technologies and Java products.
- The tips and techniques you'll learn will help you do your job better.
- You'll be exposed to cross-industry representation so you can decide which vendor best fits your needs.
- Discover which new tools are on the market today that you may need tomorrow.
- Many more technical sessions are designed for the advanced user—over 100 classes in all!
- You will have the opportunity to network with fellow software developers as well as recognized industry experts.

"An invaluable eye opener for anyone venturing into enterprise development."

Dan Rudman, Senior Software Architect, **Idetix**

Join us for four information-packed days.

Topics Covered – Take a Look at What You'll Learn

- JFC programming
- Enterprise Java platform development
- Deploying business applications with components
- Java programming tips and tricks
- Using RMI technology
- Effectively using the Java 2D API
- Database connectivity with JDBC
- Testing and debugging Java applications
- Using SWING and JFC technology
- Java-enabled data warehousing
- Tips on building applications with servlets
- Reducing memory usage in Java
- Using ORBS and CORBA with Java
- Using COM with Java
- How to use the Java sound API
- Using the Java printing API
- Adding multimedia to your Java applications
- Porting Java to the mainstream
- Exploring XML and Java
- Java Security issues
- Using mobile agents
- Java and CORBA interoperability
- Master Server-Side Java
- Understanding 3D and VRML
- Java Threads
- JavaBeans including Enterprise Beans
- Use digital signal processing efficiently
- Understand servlets

Customize Your Program.
Select from 7 concurrent tracks of courses.

- Java Primer
- Power Java
- Enterprise Java
- Distributed Java/Components
- Analysis & Design and Patterns
- Java Architectures
- Application Development

Two Day Exhibition October 19–20, 1998

Packed with leading vendors. Test drive the latest products, have your questions answered, take a sneak preview of what's new and compare what works best for you. Make time for:

- Special Events
- User Group Discussions
- Developer Days
- Product Education Sessions
- Free Product Giveaways
- Networking Opportunities
- Meet Industry Personalities

"The Road to Java" Getaway Sweepstakes

You may be one of the many lucky winners of a once-in-a-lifetime vacation getaway to Java or Bali. Just visit the show floor! We're planning a gala event packed with prizes, raffle giveaways, live entertainment and a few surprises! Look for complete details on our Website and in future mailings.

Special Keynote Address

Arthur van Hoff is Chief Technical Officer and Co-founder for Marimba, Inc., — a market leader in the intelligent distribution and management of business software applications within enterprises and across the Internet. As Marimba's CTO, Mr. van Hoff is responsible for all technical aspects of the company's product vision. Prior to cofounding Marimba, he was a Senior Staff Engineer at JavaSoft where he authored the Java compiler, the beta version of the HotJava browser, and helped design much of the Java language and the Java applet APIs.

For more information contact SIGS Conferences at: T: (212) 242-7447 • F: (212) 242-7578 • E: conferences@sigs.com • Online: www.javadevcon.com

Media Sponsors:

AltaVista · Application Development Trends · C++ REPORT · C/C++ Users Journal · Component STRATEGIES · COSMO SOFTWARE · DBMS · dejanews · developer.com · DEVX · Dr. Dobb's JOURNAL · SOFTWARE · INFOWORLD · internet.com · JAVA DEVELOPER'S JOURNAL · JAVAPro · JRO Java™ Report Online · Java SPEKTRUM · JavaWorld · JOOP · JUMBO! The Download Network · MSJ · www.sfgate.com · SOFTWARE Development · WEB DEVELOPER.COM · WEBTechniques · Windows DEVELOPER'S JOURNAL

Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems Inc. in the United States and other countries. SIGS Publications is independent of Sun Microsystems Inc.

is either derived from the Item class or that implement the IHtmlItem interface. Since HtmlBody extends the Container class, other objects that represent different HTML tags can be added to the HtmlBody object. The `print()` method can be used to print out all the HTML tags contained in the HTML page body between <BODY> and </BODY>.

Finally, the CompoundItem class brings together type-safe containers for Item objects or classes derived from Item. The methods supported by a CompoundItem are very similar to those supported by a Container. The basic difference is that a CompoundItem is also an Item, since it extends the Item class, while a Container is not.

The oracle.plsql package includes Java wrapper classes for the PL/SQL data types. There is a one-to-one mapping between these classes and the data types. Keep in mind that PL/SQL data types can have NULL values that indicate the *absence* of a value. This is different from the null keyword in Java, which signals that a reference to an object doesn't exist. These classes allow NULL values to be properly represented.

Finally, the oracle.rdbms package contains the important Session and ServerException classes. As we mentioned earlier, the Session class provides connection, and control over the connection, to an Oracle database. The ServerException class can be used extensively in writing exception-handling code (`try {} catch {} finally()`). This class includes two effective methods, getSqlcode ()and getSqlerrm(), which return the SQL error and SQL error message, respectively.

Database Access

Enterprise applications require effective database access. You can use either Java Database Connectivity (JDBC), or PL/SQL as your database access API from the Java cartridge. (If you were using PL/SQL code to leverage existing code but wanted to move to a Java implementation, you could use the `pl2java` utility to automatically generate Java wrapper classes for your PL/SQL packages and procedures.) Figure 3 shows the basic organization of an application, using the JDBC approach.

So what is JDBC? To start, it's an API specification for accessing data sources, such as Oracle, from Java applications and applets. The JDBC API is based on the X/Open Call Level Interface and the ANSI SQL-92 standard. JDBC has been widely adopted by software vendors, and JDBC drivers are available for many databases and non-database data sources. JDBC is the obvious alternative for accessing non-Oracle databases, such as Microsoft SQL Server.

If we were using the PL/SQL cartridge approach, the presentation, business, and database logic would all be implemented on the database server. With JDBC, the presentation, business and database access logic are on the application server, implemented in Java code. The business logic can

Listing 1. Generating Longhand HTML. Generating even a simple page without using any Java Web Toolkit classes uses a lot of repetitive code.

```
class HelloWorldCrude {

  public static void main(String args[])
  {
    System.out.println("Content-type: text/html");
    System.out.println("");

    System.out.println("<html>");
    System.out.println("<head><title>Crude Hello World
      Example</title></head>");
    System.out.println("<body>Crude Hello World
      example</body></html>");
  }
}
```

Listing 2. Using Oracle's Web Toolkit. This listing generates the same page using what the Toolkit calls a "component-based dynamic page construction approach". The HelloWorldWebToolkit class uses the HtmlHead, HtmlBody and HtmlPage classes from the Java Web Toolkit.

```
import oracle.html.*;

public class HelloWorldWebToolkit {
  public static void main(String args[])
  throws HtmlException {

    HtmlHead head = new HtmlHead("Hello World using Web
      Toolkit");
    HtmlBody body = new HtmlBody();
    HtmlPage page = new HtmlPage(head, body);

    body.addItem("Hello World using Web Toolkit");

    page.printHeader();
    page.print();
  }
}
```

What's Coming in OAS 4.0

Oracle Application Server 4.0 is in the works and should be released within several months. Based on a brief look at a beta copy and documentation, here's a sample of just some of the new features.

The name of the product is changing from Oracle Web Application Server to Oracle Application Server. This reflects that OAS 4.0 will support Java/IIOP/CORBA applications, referred to as JCORBA, in addition to HTML/HTTP Web applications. JCORBA allows you to easily implement business objects (components) as Java Corba Objects (JCO). These business objects define methods that clients, Java applications and Java applets, can invoke remotely. In general, these business objects do not have a user interface; they implement middle-tier business logic. Here are some of the advantages JCORBA provides:

- Management of JCORBA applications is integrated into the Oracle Application Server Manager (console) application. This provides an integrated platform for deploying both HTML/HTTP and Java/IIOP/CORBA applications.
- Hides all the Interface Definition Language (IDL) from you. You simply specify a Java interface to your objects.
- Provides centralized configuration, administration, and load-balancing features.

OAS 4.0 also includes many new features that vary by cartridge. The name of the Java cartridge has been changed to JWeb. Some of the new JWeb features include:

- The Java VM supports JDK 1.1.x.
- The JWeb cartridge is multithreaded within the same cartridge instance. Under OAS 3.0, each session-enabled Java cartridge application requires a separate Java VM process.
- Transactions can use the Java Transaction Service (JTS), in addition to transactions implemented in PL/SQL and accessed using `pl2java` wrapper classes.
- The username and browser provided to the browser are available to authenticate a request.
- A new JWeb toolkit class has been added to make intercartridge exchange requests.
- Access to configuration information via new JWeb toolkit classes.

–D.H. and S.B.

be partitioned between the application server (Java) and the database server (PL/SQL), with the choice depending on the application requirements and design. The advantages of using PL/SQL lies in the reuse of existing code; the advantages of using JDBC are that it is truly database-independent and that it is based on the pervasive Java standard.

The JDBC API is similar to the ODBC API, which has been widely adopted as a portable database access API. There are four types of JDBC drivers. Type 1 drivers, known as JDBC-ODBC bridge drivers, rely on underlying ODBC drivers. Type 2 drivers are called native API drivers, and they translate the JDBC calls into native

Listing 3. Surely Cartridge. This code generates the CaseEdit Web page using the Java cartridge.

```
import oracle.html.*;
import java.util.Date;

public class CaseEdit {

public CaseEdit() { }

static public void main(String args[]) {
  try {
  WafArgs wa = new WafArgs(args);
  String cas_id = (String) wa.get("p_cas_id");
  try { f_cas_id = Long.parseLong(cas_id); }
  catch(NumberFormatException ex) { f_cas_id = 0; }
  HdeskCase cas = null;
  WafPage wp;
  HtmlBody bd;
  String mode = null;
  if(cas_id == null) {
    mode = new String("INSERT");
    wp = new WafPage(
      "Enter new case", new Integer(3), true, null);
    bd = wp.getBody();
    wp.printHeader();

    cas = new HdeskCase();
    f_cas_id = cas.getNextCasId();

    f_cas_desc = new String("Enter short description here");
    f_cas_text = new String("Enter case text here");
    f_cas_status = new String("NEW");
    f_csd_date = new WafDate(new Date()).toStringDDMONYR();

  } else {
    mode = new String("UPDATE");
    wp = new WafPage(
      "Edit case " + cas_id, new Integer(3), true, null);
    bd = wp.getBody();
    wp.printHeader();

    // Construct "Case" object for an existing case
    try {
    cas = new HdeskCase(f_cas_id);
    f_cas_desc = cas.getCasDesc();
    f_cas_text = cas.getCasText();
    f_cas_status = cas.getCasStatus();
    f_csd_date = cas.getStatusDateString();
    }
    catch(Exception ex) {
    WafUtil.DebugLog("Exception in HdeskCase constructor");
    throw ex;
    }
  }

  bd.addItem(SimpleItem.Paragraph);
  WafForm form = new WafForm("POST", "CaseEditDispatch");

  DynamicTable tab = new DynamicTable(2);
  TableRow row = new TableRow();
  row.addCell(new TableDataCell(
      IVAlign.TOP, IHAlign.RIGHT, 1, 1, false,
    new SimpleItem("Case No:")))
     .addCell(new Long(f_cas_id).toString());
  tab.addRow(row);

  row = new TableRow();
  row.addCell(new TableDataCell(
      IVAlign.TOP, IHAlign.RIGHT, 1, 1, false,
    new SimpleItem("Case Status:")))
     .addCell(f_cas_status);
  tab.addRow(row);

  row = new TableRow();
  row.addCell(new TableDataCell(
      IVAlign.TOP, IHAlign.RIGHT, 1, 1, false,
    new SimpleItem("Status Date:")))
     .addCell(f_csd_date);
  tab.addRow(row);

  row = new TableRow();
  // TextField v_cas_desc = new TextField(
  // "p_cas_desc", 55, 80, f_cas_desc);
  // 2.1 BUG:  This generates:
  // <INPUT NAME=p_cas_desc
  // MAXLENGTH=80 SIZE=55 VALUE=Initial Value>
  row.addCell(new TableDataCell(
    IVAlign.TOP, IHAlign.RIGHT, 1, 1, false,
    new SimpleItem("Subject:")))
     .addCell("<INPUT NAME=p_cas_desc SIZE=55
        MAXLENGTH=80 VALUE=\"" +
      f_cas_desc + "\">\n");
  tab.addRow(row);

  row = new TableRow();
  TextArea v_cas_text = new TextArea(
    "p_cas_text", 60, 10, f_cas_text);
  row.addCell(new TableDataCell(
    IVAlign.TOP, IHAlign.RIGHT, 1, 1, false,
    new SimpleItem("Description:")))
     .addCell(new TableDataCell(v_cas_text));
  tab.addRow(row);

  form.addItem(tab);
  form.addItem(SimpleItem.Paragraph);
  // form.addItem(new Submit("p_save","  Save  "));
  form.addItem("<INPUT TYPE=IMAGE
    NAME=p_save SRC=\"/images/save.gif\" BORDER=0>\n");
  form.addItem(WafUtil.spacer(20));
  form.addItem("<INPUT TYPE=IMAGE
    NAME=p_cancel SRC=\"/images/cancel.gif\" BORDER=0>\n");

  // Add HIDDEN fields to pass to CaseEditDispatch
  form.addItem(new Hidden(
    "p_ses_id", (String) wa.get("p_ses_id")));
  form.addItem(new Hidden(
    "p_usr_id", (String) wa.get("p_usr_id")));
  form.addItem(new Hidden(
    "p_cas_id", new Long(f_cas_id).toString()));
  form.addItem(new Hidden("p_mode", mode));
  form.addItem(new Hidden("p_cas_status", f_cas_status));
  form.addItem(new Hidden("p_csd_date", f_csd_date));

  bd.addItem(form.toHTML());

  wp.addFooter();
  wp.print();
  }
  catch(Exception ex) {
  WafUtil.FatalError("Exception in CaseEdit",
     "Exception: " + ex.getMessage());
  }
}

  // Private fields
  private static long   f_cas_id;
  private static String f_cas_desc;
  private static String f_cas_text;
  private static String f_cas_status;
  private static String f_csd_date;

}
```

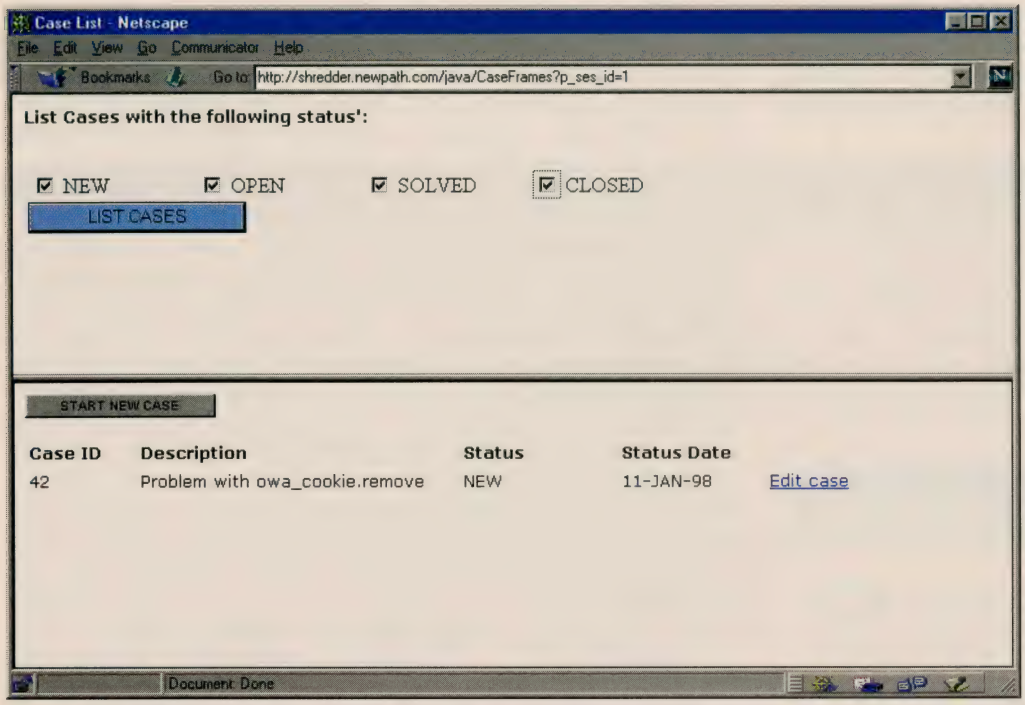


Figure 4. The HelpDesk CaseFrames Page. Our sample application uses two horizontal HTML frames to provide the main navigation and control functions.

library database calls. Type 3 drivers are net-protocol all-Java drivers, and they translate JDBC calls into a database-specific protocol by a middleware server. Finally, Type 4 drivers are native-protocol all-Java drivers, and they translate JDBC calls directly into the database-specific network protocol. Oracle supplies driver types 2 and 4 for Oracle8.

HelpDesk Sample Application

Now it's time to bring the concepts we've described together in a simple but useful application development example. In this section we'll discuss how to use the key features of the Java Web Toolkit to write a real-world business application. For our example we'll use a HelpDesk application, whose function is to track problems or cases. The application will interact with an Oracle database.

The source code for our HelpDesk application, including the database schema and all the Java cartridge code is large. So we won't show all the code here. The complete source code for our HelpDesk application can be downloaded from the Java Pro site, www.java-pro.com. (HelpDesk and other sample applications are also on the CD that accompanies our book.)

The database schema uses six tables: CASES, CODES, HACKERS, SESSION, SOLUTIONS, and USERS. The application lets users track and create new cases. Three database tables support the core of the application: USERS, SESSIONS, and CASES.

The SESSIONS table stores information for each interactive session. Since HTTP is stateless, the developer has to implement the session and state management scheme. In our example, a new session ID will be generated after the user logs on to the application by providing a valid username and password. This session ID will be passed from one page to the next by encoding the session ID into the QUERY_STRING portion of the URL as name=value pairs.

The USERS table stores information for registered users. There is a one-to-many relationship between the USERS and SESSIONS tables. A user registers by clicking on the Register button on the login page. In our Java implementation, the user has to provide a username and password each time he or she registers.

The two most important application pages, CaseFrame and CaseEdit, are shown in Figures 4 and 5.

The case list page uses two horizontal HTML frames that allow the user to list cases whose status matches the status checkbox criteria. The case status selection criteria are modified in the upper frame. The List Cases button applies the status selection criteria and generates a new list in the lower frame. Finally, the Start New Case button in the lower frame lets the user create a new case, as shown in Figure 5, while the Edit case hyperlink lets the user edit an existing case record.

The CaseEdit page allows the user to enter and edit case information. The application also includes a LoginPage, and a user RegisterPage, which are not reproduced here.

Taken together, these pages are a simple yet complete HelpDesk application that lets users create, edit, and track their cases. The application interacts with the Oracle database through

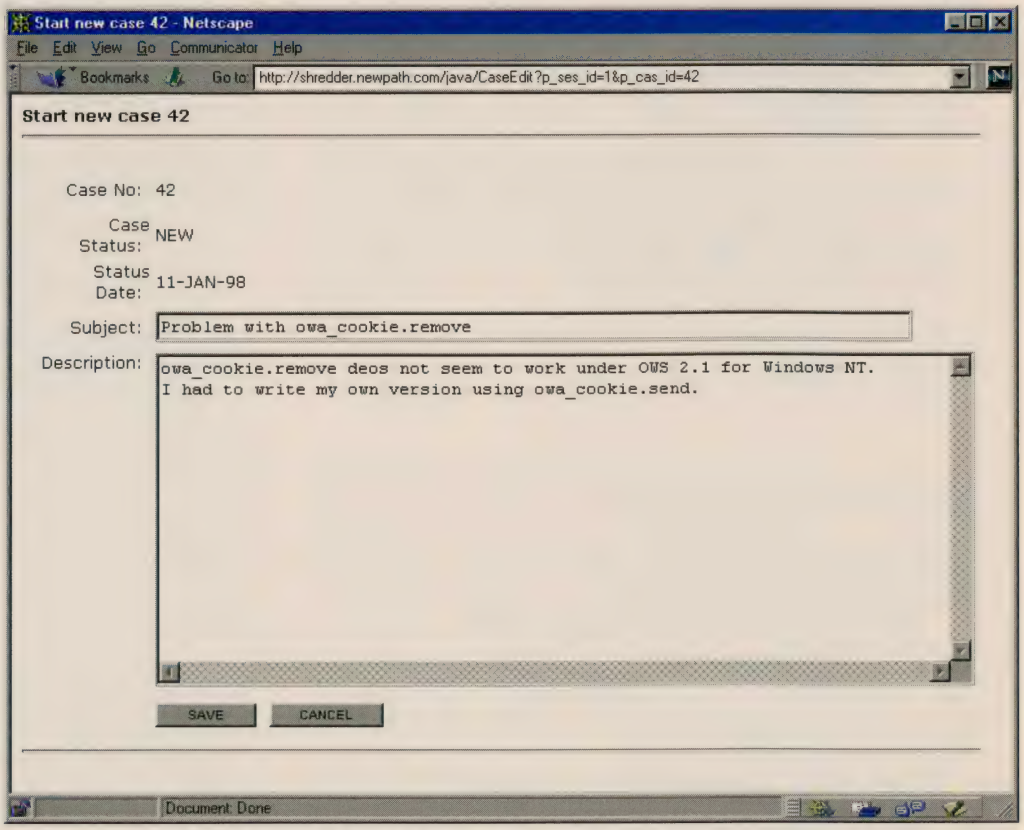

Start new case 42 - Netscape
File Edit View Go Communicator Help
Bookmarks Go to: http://shredder.newpath.com/java/CaseEdit?p_ses_id=1&p_cas_id=42
Start new case 42
Case No: 42
Case Status: NEW
Status Date: 11-JAN-98
Subject: Problem with owa_cookie.remove
Description:
owa_cookie.remove deos not seem to work under OWS 2.1 for Windows NT.
I had to write my own version using owa_cookie.send.
SAVE
CANCEL
Document: Done

Figure 5. The HelpDesk CaseEdit Page. It looks like ordinary HTML, but the façade hides a design that can shift database access methods or even database engines without affecting other layers of the application.

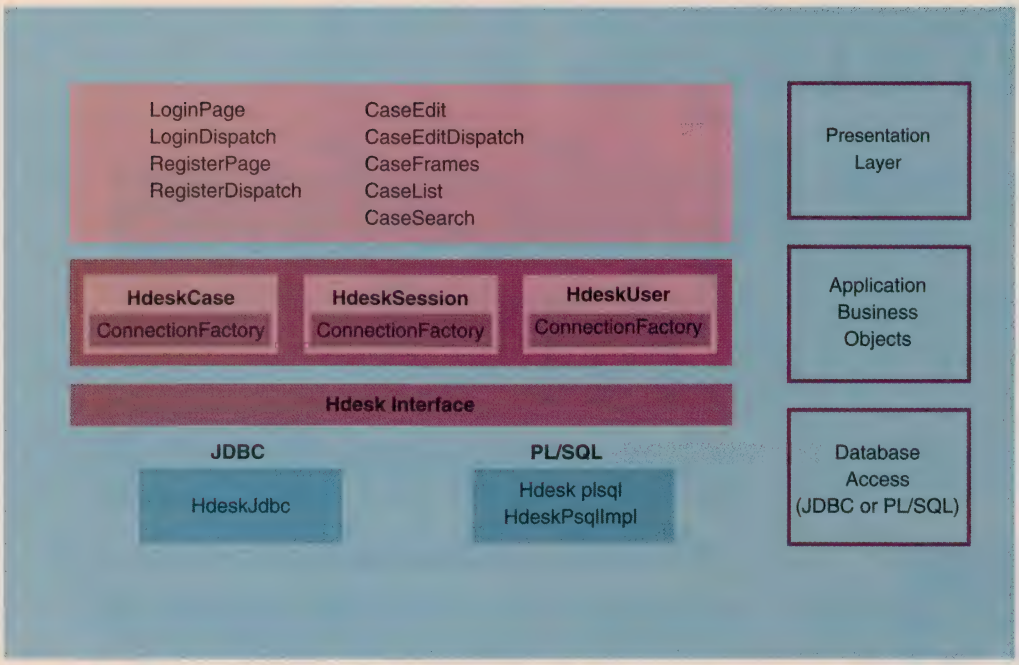


Figure 6. The HelpDesk Java Classes. The presentation layer is clearly separated from the business objects and the database details. One aspect of the distributed design is support for both JDBC and PL/SQL access in the same application.

either PL/SQL or JDBC. The database access method can be changed at run time without recompiling the application, a capability that offers significant benefits:

- Reusable classes and design constructs let developers easily implement both PL/SQL and JDBC database access.
- The need to support both methods of database access requires developers to provide clean interfaces among the presentation, business-object, and database access layers of the application.
- JDBC access to non-Oracle data sources is limited only by the availability of JDBC drivers.

In our HelpDesk design we have isolated the details of PL/SQL versus JDBC database access within the database access layer. This allows us to make changes or even change database access methods or database engines in the future without affecting the other layers. This also makes it easier for teams to work concurrently on different layers because there aren't interdependencies between the layers. Figure 6 shows how our Java classes are associated with the presentation, business-object, and database access layers.

Although we don't have the space to review all of the HelpDesk code, Listing 3, the source code for the CaseEdit page, should give you a feel for Java cartridge programming.

That concludes our whirlwind tour of Oracle Web Application Server. What's exciting about OAS is that it removes almost all the software (and associated costs) from the client, except for the Web browser. We rely instead on the HTTP request-response protocol to provide an RPC-type service to the application server.

This ultra–thin client approach also slims down deployment and support costs, no small feat in today's cost-conscious corporate world. In addition, software distribution costs are practically gone as well, since the HTTP protocol allows the user-interface application, HTML, and other MIME types to be distributed dynamically from the application server.

But perhaps the biggest benefit is that our HTML/HTTP applications can be accessed and used by virtually anyone, no matter where they "reside" on the network. Apps are no longer confined to licensed users within the walls of a single company. The implications of this model for new ways of conducting business—indeed, for the creation of whole new businesses—are exciting. They also drive lawyers crazy, which indicates that we must be on to something big! ■

Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

Plugging into Oracle
Locator+ Code: JP0898DH

TechTIP

Some Objects Are More Equal Than Others

A Vector is a convenient tool for keeping track of a set of objects, particularly when you don't know in advance how many will be in the set. You can find out whether an object is present in the set using one of three methods: `contains()`, `indexOf()`, and `lastIndexOf()`. But all three use the object's equals method to compare it with the members of the Vector. If the object does not have an equals method, and relies on the default built into the Object class, you'll be searching for another reference to the same object, instead of an object with the same characteristics. To illustrate this, suppose you have a Vector of Buttons:

```
Button okBut = new Button("OK");
boolean okHere =
  yourVector.contains(okBut);
```

The boolean will always be returned false because the new Button cannot be in the Vector already. Even if there is an existing Button in the Vector with the same caption, it will not be reported, because the Button class inherits its equals method from the Object class. Now suppose you have a Vector of Points:

```
Point origin = new Point(0, 0);
boolean datum =
  yourVector.contains(origin);
```

This boolean will return true if there is a Point in the Vector with the same coordinates, because the Point class has an equals method which compares the x-and y-values of the two Points. Conversely, if you have an existing Point and you want to check whether it is already referenced in the Vector, you need to work a little harder:

```
boolean pointHere = false;
for (int i = 0 ; i < yourVector.size()
  ; i = i + 1) {
  if (yourVector.elementAt(i) ==
    yourPoint) {
    pointHere = true;
    break;
  }
}
```

Note that a Vector's `removeElement` method also relies on the functionality of the object's equals method.

—Tom Yarker

Build **Strategic Enterprise Applications** on Windows NT

Dallas

November 16/17

NT Enterprise Developer's Conference is a new technical conference brought to you by the organizers of the VBITS conferences and publishers of *Visual Basic Programmer's Journal*.

NT Enterprise Developer's Conference is two days of coordinated sessions taking you through the most important tasks developers must accomplish to create effective, scalable, client/server applications on Windows NT. You'll see NT-based technologies from IBM and Microsoft and work through designing, building, testing, debugging and deploying component-based solutions. You'll learn practical approaches to data access to SQL Server, Oracle and legacy systems. You'll understand how technologies such as MTS, ADO, RDO, COM, DCOM, IIS, ASP, ODBC and OLE DB fit together to extend client/server development on NT across the entire enterprise.

Technical Content and Real World NT Enterprise Developer Solutions

Like VBITS conferences, NT Enterprise Developer's Conference is technical, aimed at professional programmers. You'll see plenty of code and practical real-world techniques, and you'll hear about real-world solutions from a faculty of professional providers who have implemented them for corporate clients.

Presented By FTP FAWCETTE TECHNICAL PUBLICATIONS
Publishers of *Visual Basic Programmer's Journal*, *Java™Pro*, and VBITS Conferences
209 Hamilton Ave., Palo Alto, CA 94301-2500
Tel: 650-833-7100, Fax: 650-321-3818

Microsoft, Windows, Windows NT, Visual Basic, and Visual C++ are registered trademarks of Microsoft Corporation.
Microsoft Transaction Server is a trademark of Microsoft Corporation. IBM is a trademark of IBM Corporation. Java is a registered trademark of Sun Microsystems, Inc.
VBITS and NT Enterprise Developer's Conference are trademarks of Fawcette Technical Publications.

Atlanta image copyright © 1998 PhotoDisc, Inc. Dallas image courtesy of Dallas Convention & Visitors Bureau.

Enterprise Developer's Conference '98

Dallas Conference Agenda

Day One

Building Client/Server Applications: Parts I and II
Implement a component-based architecture on NT in the real world.

The Data Services Layer
Design and implement the data services layer of your component-based application.

Access to Oracle and Legacy Databases
Access Oracle databases with COM components, ODBC, and MTS, plus legacy DBMS access.

Designing Components for Microsoft Transaction Server
Implement ActiveX Server DLLs under MTS control. Examples in VB, C++, Java and Delphi.

Testing and Debugging Component-Based Solutions
Test and debug a distributed system, including components and encapsulated unit testing.

Evening Panel Discussion: Managing Component-Based Solutions Development
A panel for project managers dealing with distributed design, development and testing.

Day Two

Essential COM
Understand COM's features in order to create more effective COM and ActiveX components.

Hard-Core DCOM
Use DCOM to distribute your application, while improving performance and security.

Deploying Component-Based Solutions
Use the Web and custom installation code to deploy your applications.

Programming with Middleware on Windows NT
Choose among competitive programming environments for NT development.

Integrating MSMQ and ASP into MTS Applications
Build reliable asynchronous communications channels between high-volume server apps.

Leveraging Your Component-Based Solutions to the Intranet or Internet
Link Web access to DCOM and MTS-based components, and Web-deploy n-tier apps.

Register by October 5 and SAVE $100

Call for information on corporate discounts

1-800-848-5523

Outside the U.S. & Canada call 1-650-833-7100

Register online at www.vbits98.com

Eventfully Speaking

There are many ways to write event-driven programs, but how do you write programs that are flexible and fast?

by Larry O'Brien

There are as many ways to write an event-driven program as there are to skin a cat. Java has already tried two: one model for the 1.0 language release and another for the 1.1 and future releases (see Figure 1). Both attempt to tackle one basic question: How do you architect a program to be both flexible and fast?

During a recent garage cleaning, I stumbled across a printout of my first commercial program (thus establishing myself as somewhat of a newbie—if I were an old-timer, I would have stumbled across a papertape or card-deck). There, in faded ink between the sprocket holes of the line printer, was an interface to make any programmer proud:

```
60  PRINT "HOW MANY TEETH";
70  INPUT T
80  PRINT "HOW MANY
    POINTS";
90  INPUT P
```

The program would then cut a saw on a numerically controlled laser cutting machine (exactly how, I can no longer tell, despite such crystalline lines as `260 REM I WAS HERE BUT NOW I'M GONE`). The thought of a "flexible" program, one that treated the user as anything other than an operator who provided the specified input, was strictly outside the pale. Now, the pendulum has swung to the other extreme. At a recent conference, I attended an excellent lecture titled "The Interface *Is* the Game." Whereas once the goal was strictly to get the job done with as little consumption of computer resources as possible, now a major goal of program design is to minimize modality (where your program demands a particu-

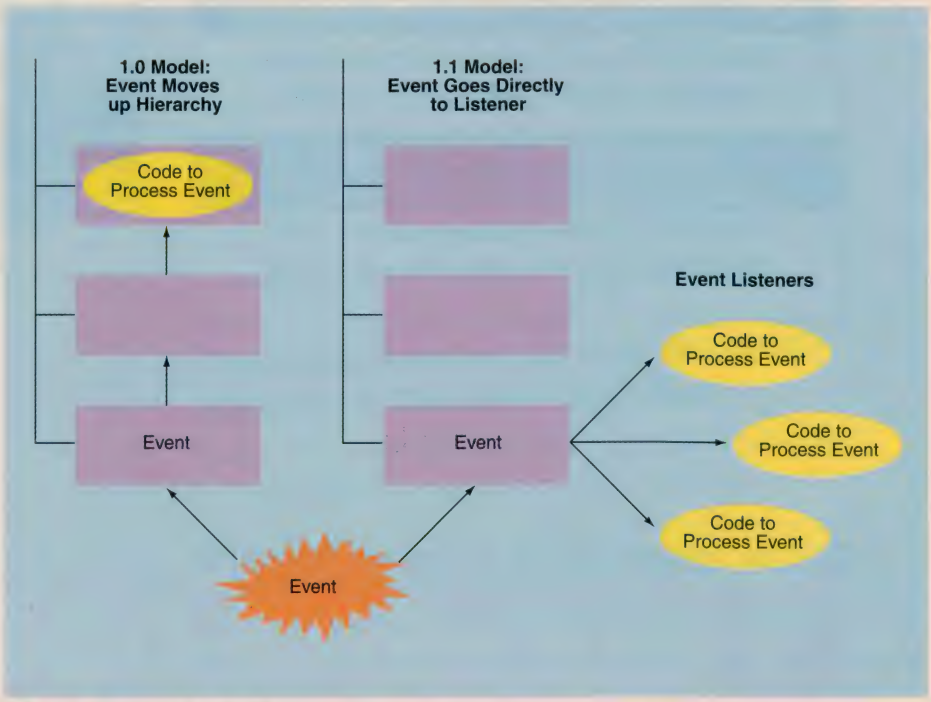


Figure 1. Listening for Events. In the Java 1.0 model, events are propagated up the hierarchy, whereas in the Java 1.1 model, objects can "listen" for events in another object. The source object broadcasts its event directly to all listeners.

Larry O'Brien is an independent trainer and consultant and developed the *Bruce Eckel's Hands-On Java CD*, available from www.mindview.net. A well-known voice in the development community, he is the chair of the "Internet and Intranet Development" track for the Software Development conference. Reach him at larryo@wco.com.

Listing 1. **Procedural Legacy.** After the initial interface set up, this simple applet reveals evidence of the classic event queue in the handleEvent() method. While the actual event queue may not be apparent, you can clearly see the legacy of the procedural code.

```
import java.awt.*;
import java.applet.*;

class AWT1Events extends Applet{
  Button show = new Button("Show");
  Button set = new Button("Set");
  TextField fldTeeth = new TextField("10");
  TextField fldPoints = new TextField("5");

  int teeth, points;

  public void init(){
    setLayout(new FlowLayout());
    add(show);
    add(set);
    add(fldTeeth);
    add(fldPoints);
  }

  public boolean handleEvent(Event evt){
    if(evt.id == Event.KEY_RELEASE){
      this.getAppletContext().showStatus("Key typed!");
    }
    return super.handleEvent(evt);
  }

  public boolean action(Event evt, Object arg){
    boolean handledHere = false;
    if(evt.target.equals(show)){
      this.getAppletContext().showStatus("Teeth: " + teeth + "
        Points: " + points);
      handledHere = true;
    }
    if(evt.target.equals(set)){
      teeth = Integer.parseInt(fldTeeth.getText());
      points = Integer.parseInt(fldPoints.getText());
      handledHere = true;
    }
    if(handledHere != true){
      this.getAppletContext().showStatus("Passing \"up\" event: "
        + evt + " with argument " + arg);
      handledHere = super.action(evt, arg);
    }
    return handledHere;
  }

}
```

lar sequence of inputs). Today, a program should be able to respond to arbitrary mouseclicks, menu selections, network events, and commands from input devices hardly contemplated by the programmer, such as vision- and head-tracking systems. At the same time, one of the great lessons learned in the personal computer era is that responsiveness is essential: the program must never freeze, it should respond instantly to all input.

I'm afraid that my old BASIC interpreter, with its "stop-and-ask-for-input" model, wouldn't have measured up. However, at about the same time I wrote my saw-cutting program, the smart people at PARC were perfecting a flexible and fast architecture that would turn the computing world on its head. Or at least turn programs inside out. The event-driven graphical user-interface architecture proposed that input should be the top priority. Every time the user types a key or moves an input device, information about that "event" should be placed in a queue. If no such events were being generated, the program would examine the front of the queue and take appropriate action (this was strictly a secondary priority—you could often bog down early GUI programs by wiggling the mouse back and forth). One of the most important concepts was that the single queue contained all events being sent to the program, not just those that the program expected, or those of a specific type. Keyboard punches, mouse moves, and joystick waggles were told to stand in the same line and wait their turn. Additionally, if a mouse clicked on a button or if the Enter key was hit while a button had focus, a so-called semantic event would also be added to the queue (usually in addition to, but sometimes replacing the original physical or low-level event).

But a keypress and a mouseclick are very different things, with very different types of data. Instead of placing the raw data of an event in the queue, what was actually placed was a constant identifying the type of the event and a pointer to a memory location containing the event's data. Since the size of the constant the size of a pointer are always the same, this makes the event queue a straightforward data structure. It is this data structure at the heart of the Java 1.0 event model. Take a look at Listing 1, which shows a very simple applet to replace my BASIC code.

After some minimal code to declare, initialize, and set up two buttons and two text fields, we see evidence of the classic event queue in the method `public boolean handleEvent(Event evt)`. While the actual event queue may not be apparent, you can clearly see the legacy of the procedural code. Instead of a constant to identify the event type and a pointer to data, the AWT 1.0 model uses an Event object. We determine if the Event was a key pressed and, if it was, we place a message on the browser's status bar. If not, we pass the Event to the `handleEvent()` method of our superclass, Applet.

The Java 1.1 event model is based on the concept of EventListeners that subscribe to a Component

But if the `handleEvent()` method is used to look at events on the queue, what's the method `public boolean action(Event evt, Object arg)` for? An Event object is created on a Component (the parent class of most AWT objects). After `handleEvent()` gets a crack at an Event, the AWT examines it and may call one of the following methods: `action()` (if the event is the "standard action" for the Component, such as a clicked event for a button), `keyDown()`, `keyUp()`, `lostFocus()`, `gotFocus()`, `mouseDown()`, `mouseUp()`, `mouseMove()`, `mouseDrag()`, `mouseEnter()`, and `mouseExit()`. In our example, we're interested in the "standard event" of the show or set buttons and so we put our code in the `action()` method.

Confused? Ironically, this model was supposed to make things easier. The hope was that most programs and programmers would deal solely with the action event. It rarely works

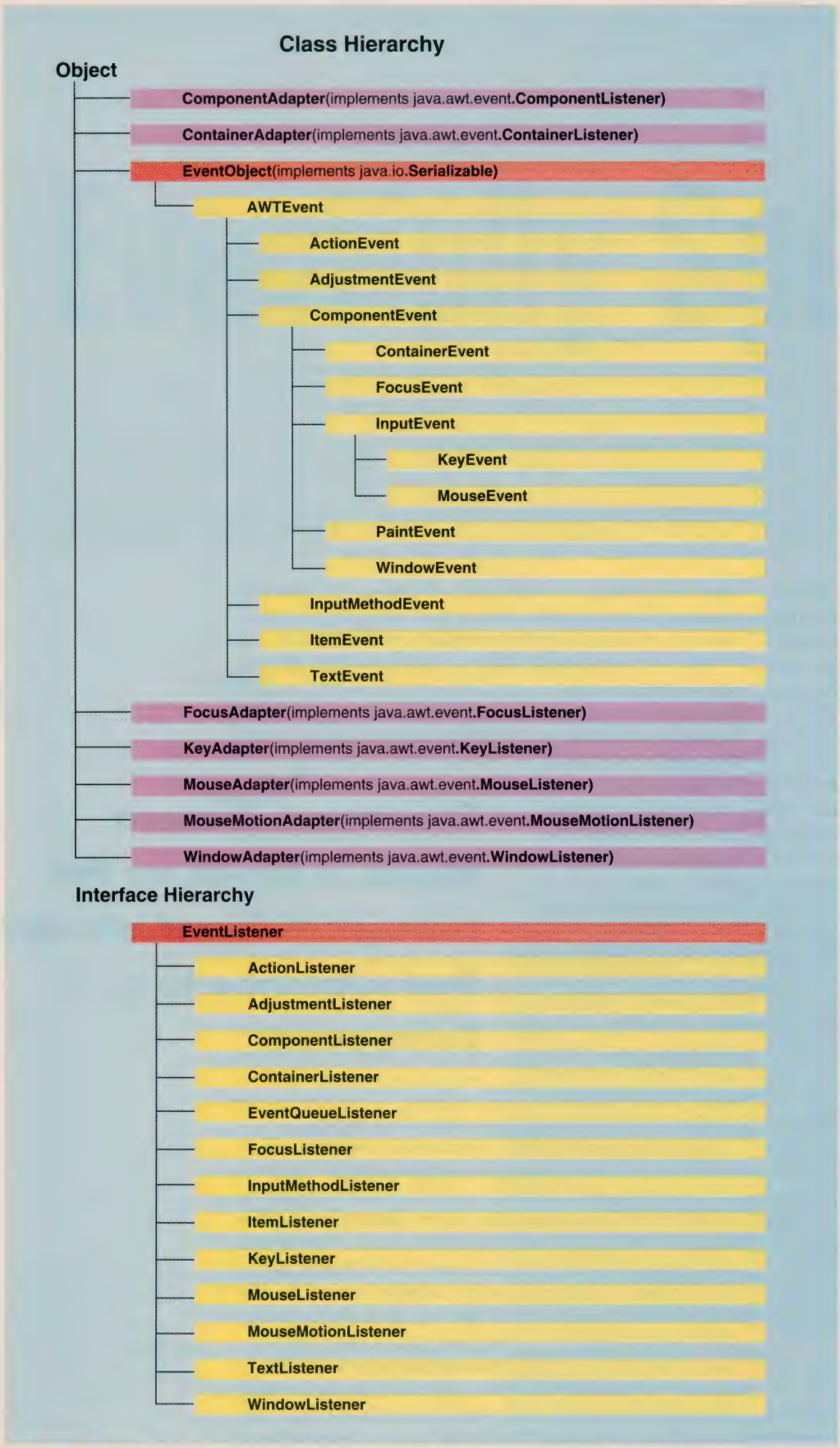


Figure 2. New Family Members. Although straightforward conceptually, the Java 1.1 model introduces several new class families. The EventListener model, with its inheritance-based Events, can be easily extended to support new Event types.

out that way, however, and if you're using AWT 1.0, you'll need to be aware of these methods.

A few things should be stated about this model. First, it's not easily extensible. Behind the scenes, Java's `java.awt.Event` class has a numeric constant that identifies what type of Event it is (for instance, a `KEY_PRESS` event is 401). Adding a new type of Event (such as `JOYSTICK_BUTTON_PRESS`) requires modifying and recompiling Event, which would get you in trouble with Sun. Second, in a large application, `action()` and `handleEvent()` could get huge. A quick count shows that the word processor I'm using has at least 100 events that can be triggered in its default state. A similar word processor in Java would require an `action()` method hundreds of lines long. That might make both maintenance and performance tuning difficult. Third, it's a bottleneck. All events go into a single queue and are taken from the front, one at a time. In a real implementation, you can add complexity in order to minimize this (for instance, multiple events can be coalesced, or certain events can be given priority).

Despite these problems, the model works. You can write large applications based on this model of event handling. That might seem obvious, but a few years ago, many programmers doubted that a chaotic stream of events and responses was a good model for a large application. One of the biggest concerns was how to deal with lengthy calculations. One of the major selling points of the event-driven model was that it should prevent your program from freezing at any point, even if you're doing a lengthy calculation. Although there are several ways around this, Java's support for threads naturally comes into play; if you need to do a particularly CPU-intensive calculation, make it a thread.

But wait a second—if you have threads you shouldn't have to worry about the user interface freezing, you just dedicate a thread to it, right? And if you have objects, surely there's a way to leverage the combination of data, behavior, and polymorphism—that's the very definition of object-oriented programming? Indeed there is, and it's more elegant than the previous model of event queues.

In the event model of Java 1.1, every Component keeps a list of zero or more `EventListeners`. When an event occurs, such as a mouseclick or a keypress, the Component calls an appropriate method for each of the `EventListeners`, passing in details of the event in an instance of a subtype of `java.util.EventObject`. In other words, the Java 1.1 model is based on the simple concept of `EventListeners` who subscribe to a Component and, when an event happens, the Component publishes the information to its subscribers. This design pattern is sometimes called "Observer" and sometimes "Publish-Subscribe."

Although straightforward conceptually, the Java 1.1 model introduces several new class fami-

Required Reading!

WEB Builder

The built-to-last independent resource for professional Web developers and designers

Only $27.95 for 12 monthly issues!

It's for Web Developers

WEB Builder is the independent source that gives you the highly technical, code-intensive articles you need to build Internet/intranet applications and Web sites better and faster.

It's for Web Designers

WEB Builder extensively covers all that goes into designing interfaces for sophisticated applications—from application architecture to Web site graphics.

It's Comprehensive

From in-depth product reviews and high-caliber technical features to loads of valuable code, *WEB Builder* is the comprehensive tool for the new corporate Web development team.

Make *WEB Builder* Your Source for Technical Information

Order today and you'll get 12 issues of *WEB Builder* for only $27.95. You'll enjoy delivery straight to your mailbox, have Web access to every issue's code—and you'll save over 60% off the annual newsstand cover price!

ORDER TODAY!

Complete the subscription card included in this magazine and drop it in today's mail.

Or call 1.800.848.5523

To subscribe online:
http://www.webbuildermag.com/subscribe

FTP FAWCETTE TECHNICAL PUBLICATIONS

Basic annual subscription rate is $32.97. Savings based on annual newsstand cover price of $71.40. Subscription begins upon receipt of payment. Please allow 4 weeks for delivery of first issue. International subscriptions must be payable in U.S. dollars plus postage: Canada/Mexico add $18/year for air mail delivery; all other countries add $44/year for air mail delivery.
Visual Studio is a trademark of Microsoft Corp.

Listing 2. Longer, yet Simpler. It may look more complex, but Listing 2 is actually significantly less so than Listing 1, and it is only two lines longer.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

class AWT11Events extends Applet{
  Button show = new Button("Show");
  Button set = new Button("Set");
  TextField fldTeeth = new TextField("10");
  TextField fldPoints = new TextField("5");

  int teeth, points;

  public void init(){
    setLayout(new FlowLayout());
    add(show);
    add(set);
    add(fldTeeth);
    add(fldPoints);

    set.addActionListener(new SetPressedListener());
    show.addActionListener(new ShowPressedListener());
    MyKeyListener mkl = new MyKeyListener();
    fldTeeth.addKeyListener(mkl);
    fldPoints.addKeyListener(mkl);
  }

  class SetPressedListener implements ActionListener{
    public void actionPerformed(ActionEvent evt){
      teeth = Integer.parseInt(fldTeeth.getText());
      points = Integer.parseInt(fldPoints.getText());
    }
  }

  class ShowPressedListener implements ActionListener{
    public void actionPerformed(ActionEvent evt){
      getAppletContext().showStatus("Teeth: " + teeth + "
        Points: " + points);
    }
  }

  class MyKeyListener extends KeyAdapter{
    public void keyReleased(KeyEvent evt){
      getAppletContext().showStatus("Key pressed!");
    }
  }

}
```

lies, as listed in Figure 2. A conversion of our applet to the new model is shown in Listing 2. It may look more complex, but Listing 2 is actually significantly less so than Listing 1 (there's not a single if statement in Listing 2), and it is only two lines longer. Most importantly, the Java 1.1 `EventListener` model, with its inheritance-based Events, can be easily extended to support new Event types. If you're worried about the performance overhead of all this "object-oriented stuff," don't be. Some rough benchmarking shows that the Java 1.1 event model dispatches events faster than the 1.0 `handleEvent()` model.

First, we use the line `import java.awt.event.*` to bring in the Java 1.1 event-model classes. (The files in java.awt.event import the java.util classes—it's not necessary to explicitly import them). The basics of setting up the fields and laying them out remain the same, but halfway through `init()`, things change. We have to take on the initial duty of "wiring up" the Components with their various `EventListeners`. The buttons Set and Show get new instances of a `SetPressedListener()` and a `ShowPressedListener()` (which we'll define a few lines later). The textfields share a single instance, `mkl`, of the `MyKeyListener` class.

Next we define the `EventListeners`. Notice that we're using "inner classes"—the definition of `SetPressedListener()` and the other Listeners is done within the enclosing scope of the AWT11Events class. This isn't strictly necessary; any accessible class that implements the appropriate EventListener interface can listen to a Component. The `ActionListener` interface implemented by `SetPressedListener()` and `ShowPressedListener()` has a single method `actionPerformed()`, in which we implement our functionality.

You'll notice something's different about `MyKeyListener`. Instead of saying it implements `KeyListener`, it's defined as extending `KeyAdapter`. What gives? Certain interfaces derived from `EventListener` have several methods. The `KeyListener` interface, for instance, has methods `keyPressed()`, `keyReleased()`, and `keyTyped()`. A class that implements `KeyListener` has to implement all three. As an aid to lazy programmers (like me), the `xAdapter` classes provide an empty implementation of these interfaces. By extending an adapter class you can override the methods of interest to you.

How does the Java 1.1 `EventListener` model compare with the Java 1.0 `handleEvent()` model? First, it's more extensible. The events are distinguished by an inheritance relationship, therefore extending the model doesn't require recompiling a monolithic Event object. It simply requires that a new Event type and an associated Listener type be declared to extend the hierarchy as appropriate. Second, it hides the complexities of dispatching the event to the appropriate target object, thus eliminating the need for lengthy `handleEvent()` and `action()` methods. Third, it trades program complexity for design complexity. You need to create a new subtype of `EventListener` for every behavior you want to program. Our theoretical word processor, with its hundred events, might have to create a hundred different subtypes, each of which has a single instance, and each of which listens for a single event.

Or would it? In our example, we see that we're sharing a single instance of `MyKeyListener`. If we need more flexibility, we could save state in either multiple instances of the `EventListener` sub-type or, if we want to trade off additional complexity for less memory use, we can use the Flyweight design pattern. Listing 3 shows examples of both approaches.

The two approaches are embodied in the Panel sub-types we declare, `StateInListener` and `StateExternal`. Each sub-type does the same thing; create a grid of buttons with random numbers and increment a button's number every time it is clicked. However, the `StateInListener` creates a new `StateListener` for every button, while the `StateExternal` class creates a single `FlyweightListener` that is shared between buttons.

The `StateInListener` is simpler. Each button has an associated instance of a `StateListener`, every `StateListener` has a handle to its button and an integer value, which it increments when `actionPerformed` is called. This is extremely straightforward, but requires an instance of `StateListener` for every button.

If you were trying to minimize memory requirements, you might prefer the `StateExternal` approach. The `FlyweightListener` relies on state being stored externally—in this case the current value of a number is "stored" in the label of the button that was pressed. Which button was pressed? That information is stored in the `EventObject`, and we retrieve it using the `getSource()` method. Although we've gained memory, we've increased the amount of processing for every button press (retrieve the source, retrieve the label, and convert it to an integer—all in order to save a handful of memory) and we've decreased the coherence of the `FlyweightListener`. In general, because of these tradeoffs, the Flyweight pattern is only a good idea when either memory conditions are extremely tight or if you're talking about a situation where a single or a handful of Flyweights can stand in for thousands of instances.

One objection that might be raised to the Java 1.1 event model is that in a large program, one might have dozens of small `EventListener` classes doing mundane chores and cluttering the source code. This is where Java's anonymous

inner classes can come into play. Given a Button b, the following code is perfectly legal:

```
b.addActionListener(new ActionListener(){
  public void actionPerformed(
    ActionEvent evt){
    getAppletContext().showStatus(
      "Button pushed!");
  }
});
```

This code might be somewhat startling. A call to a constructor followed by a bracket actually means "Create a new anonymous subclass that extends (or implements) the named class (or interface) and which is defined in the scope following the bracket." Although the combination of brackets and parentheses can be intimidating at first, one's fingers quickly learn the sequence. I use anonymous inner classes unless the `EventListener` is complex or shared.

Microsoft claims that inner classes and anonymous inner classes are confusing and wasteful. This is their major justification in introducing the "bound method reference" model in the upcoming Visual J++ 6.0 (VJ++ supports both the classic event models, but not in its visual designer). While the syntax of anonymous inner classes might not be intuitive, neither is it particularly challenging, nor are the semantics straightforward. The wastefulness criticism might be more valid, since every `EventListener` class that is created is, well, a class. Most casual programmers, and all GUI builders that I've seen, create a new `EventListener` class for every component, which will result in a separate class file for each class (and therefore an extra server hit unless JAR or ZIP files are used), class loading and perhaps validation, and an increased working set. However, the techniques shown in Listing 3 (multiple instances of an `EventListener` class and the Flyweight pattern) should help increase efficiency in applications large enough for this to be an issue.

Throughout this article, we've coded the response to an event directly into the action or handleEvent() method (Java 1.0 model), or directly into the EventListener's `xPerformed()` method (Java 1.1 model). This simplifies the code, but it's not the correct way to structure event handling for a larger system. Rather, a large system should use EventListeners to convert events into domain service requests, such as "book ordered" or "change of address" One or more domain classes will typically use the Observer pattern to alert various visual interface classes to update themselves. (You'll often see this inaccurately called the Model-View-Controller architecture.) The end result is an object-oriented event model that is fast, responsive, consistent, extensible, and very scalable.

The `EventListener` model is easier to use, more modular, and more extensible than the Java 1.0 event model. Because of these advantages, as well as the confusion inherent in having two different event models, the Java 1.0 event model is deprecated. Unfortunately, the `handleEvent()` model is virtually your only choice if you want to program applets for delivery over the Internet; it appears that the majority of those browsers that support Java support only Java 1.0. Unfortunately, the statistics at BrowserWatch are insufficient to be sure. This will, of course, change over time. Internet Explorer 4 supports the Java 1.1 event model, as does Netscape Navigator 4.05, which is in beta at the time of writing.

I don't know a portable way to use JavaScript to determine the version of Java supported by the browser, but `System.getProperties("java.version")` will return a String containing the version. You could use this value to load a 1.1-based applet if possible, and an older-styled applet otherwise.

For now, however, one must be familiar with both Java event models. Perhaps it is best to approach it as a lesson in object-oriented design. By increasing the level of abstraction and using polymorphism to select behavior, a design can become simpler, more elegant, and even speedier. Doing more with less is the hallmark of a successful design, and the Java 1.1 event model is a perfect example. ■

Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

Eventfully Speaking
Locator+ Code: JP0898LO

Listing 3. Flyweight Design Pattern. Two approaches are embodied in the Panel sub-types we declare, StateInListener and StateExternal. Each sub-type does the same thing; create a grid of buttons with random numbers and increment a button's number every time it is clicked. However, the StateInListener creates a new StateListener for every button, while the StateExternal class creates a single FlyweightListener that is shared between buttons.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

class ManyListeners extends Applet{
  public void init(){
    add(new StateInListener(), BorderLayout.NORTH);
    add(new StateExternal(), BorderLayout.SOUTH);
  }
}

class StateInListener extends Panel{
  StateInListener(){
    setLayout(new GridLayout(3,3));
    Button[] bArray = new Button[9];
    for(int i = 0; i < 9; i++){
      int value =(int) (Math.random() * 10.0);
      bArray[i] = new Button(Integer.toString(value));
      bArray[i].addActionListener(new StateListener(bArray[i],
value));
      add(bArray[i]);
    }
  }

  class StateListener implements ActionListener{
    Button source;
    int value;
    StateListener(Button source, int value){
      this.source = source;
      this.value = value;
    }

    public void actionPerformed(ActionEvent e){
      source.setLabel(Integer.toString(++value));
    }
  }
}

class StateExternal extends Panel{
  StateExternal(){
    setLayout(new GridLayout(3,3));
    Button[] bArray = new Button[9];
    FlyweightListener fl = new FlyweightListener();
    for(int i = 0; i < 9; i++){
      int value = (int) (Math.random() * 10.0);
      bArray[i] = new Button(Integer.toString(value));
      bArray[i].addActionListener(fl);
      add(bArray[i]);
    }
  }

  class FlyweightListener implements ActionListener{
    public void actionPerformed(ActionEvent evt){
      Button src = (Button) evt.getSource();
      int value = Integer.parseInt(src.getLabel());
      value++;
      src.setLabel(Integer.toString(value));
    }
  }
}
```

Threading 101

Add multithreading to your next application for increased performance

by Luke Andrew Cassady-Dorion

Since the beginning of the Java craze, programmers around the world have developed a fond attachment to the language. In Silicon Valley, you'd have to search high and low to find anyone with a real distaste for Java. Programming in Java allows developers to concentrate on implementing business logic, instead of silly things like memory management and dangling pointers. In developing Java, the engineers at Sun made a conscious decision to make development easy and include the features necessary to create modern applications. Back when applications were executed from the DOS prompt, we needed to do little more than crunch some numbers or draw some primitives to the screen. Modern applications, however, are likely to perform some network transactions, contain an evolved GUI, and are packed with features. A major concern when providing a feature-packed application is speed, because we often ask the processor to perform more than one task at a time. To address this issue, Java contains native support for a technology called "threads".

A thread, in essence, allows multiple blocks of code to execute at the same time. Even though a single processor machine can only execute a single line of code at a time, with threads it is possible to rapidly switch between code blocks to give the appearance of simultaneous execution. As an example, consider a Web browser application. When a Web page is loaded, a large number of files need to be obtained. The HTML content is at least one file, each image has its own file, and each multimedia element is still another file. Instead of loading each file in sequence and displaying only when ready, the Web browser will obtain each file in a unique thread and dedicates a thread to displaying files as they arrive. Even though the files do not arrive sooner through the use of threads, it appears to the user as though they do.

Figure 1. Without Threads. In the non-threaded version of our application, only one square changes color.

While threading is one of the more exciting features in Java, it may also be one of the most under utilized. Many programmers fear working with threads, because they can be confusing and create opportunities for bugs to rear their ugly heads. However, as a developer you must realize that by making your application multithreaded, you make it faster and increase your ability to beat the competition. In this article, we'll examine the qualities that make up a thread, the concerns facing developers using threads, and write a few applications demonstrating the use of threads.

Qualities of a Thread

As I stated earlier, threads allow separate blocks of code to execute at the same time. This is accomplished by identifying all code that needs to execute and giving each block a turn at the processor. In general, execution turns are distributed in a democratic manner;

Luke Cassady-Dorion, a Sun Certified Java 1.1 Programmer, is a software engineer at Metis LLC (www.metisllc.com). With the aid of various distributed-object technologies, including CORBA, Luke has deployed many enterprise-wide solutions written entirely in Java. Luke is the author of numerous articles on distributed computing and Java; he recently published the book *Industrial Strength Java* (Indianapolis, IN: New Riders Publishing, 1997). Reach him at luke@luke.org.

each block gets an even turn and no one thread gets preferential treatment over another. While an even resource distribution may work in most situations, there are times when one or more threads will need to execute before others. In these situations, it is possible to give each thread a priority that determines its execution order. When there are threads of different priorities, resources are first shared between all threads with the highest priority and then with threads at each of the lower priorities.

Concerns Facing the Thread Developer

Threading undoubtedly makes an application better, but you must not forget that control flow can't be assumed. If two threads share access to a common variable, both must realize that the value of that variable may be changed by another thread. To prevent this from causing problems in your application, shared resources must be locked before they are accessed. For example, in a banking application, when a withdrawal transaction occurs, the normal process is to first check the account balance and then either make a withdrawal or report insufficient funds. However, if two threads access the account at the same time, confusion could occur. Consider the following process:

1. Thread A checks the account balance and sees that there is $100.
2. Thread B checks the account balance and sees that there is $100.
3. Thread A withdraws $75.
4. Thread B withdraws $75.
5. The account is overdrawn.

Listing 1. A Threading Demonstration. The first class, ThreadDemo, extends Thread and prints its logical name. The second class, ThreadRunner, instantiates two ThreadDemo Objects and starts them running at the same time.

```
// ThreadDemo.java

/**
 * Simple thread demo
 */
public final class ThreadDemo extends
Thread {
   private String _sName;
t
   public ThreadDemo(String sName) {
      _sName = sName;
   }

   public void run() {
      for(int i=0; i<50; i++) {
         System.out.println(_sName+": "+i);
      }
   }
}

//ThreadRunner.java

list
list list
   tab indents
   not spaces
   turnovers should indent one
   additional tab.
   tab indents
list
list list
```

In this example, access to the account was not locked for the duration of the transaction and, as a result, the account become overdrawn. Consider this alternative process utilizing proper locking:

1. Thread A locks access to a bank account.
2. Thread B attempts to access the bank account and is denied.
3. Thread A checks the account balance and sees that there is $100.
4. Thread B attempts to access the bank account and is denied.
5. Thread A withdraws $75.
6. Thread B attempts to access the bank account and is denied.
7. Thread A releases the lock on the account.
8. Thread B locks access to the bank account.
9. Thread B checks the account balance and sees that there is $25.
10. Thread B reports that it is not possible to withdraw the required $75.

Figure 2. Threads at Work. In the threaded version of our application, all squares change color.

As these steps show, by locking access to the shared account resource, the unfortunate overdraft situation is avoided. It's easy to see how major problems could arise if shared resources are not properly locked.

At this point, you should have a pretty good understanding of why threads exist and the issues developers working with threads face. The rest of this article will be spent learning how threads are implemented using Java. We'll also compare performance in a threaded and non-threaded version of the same application.

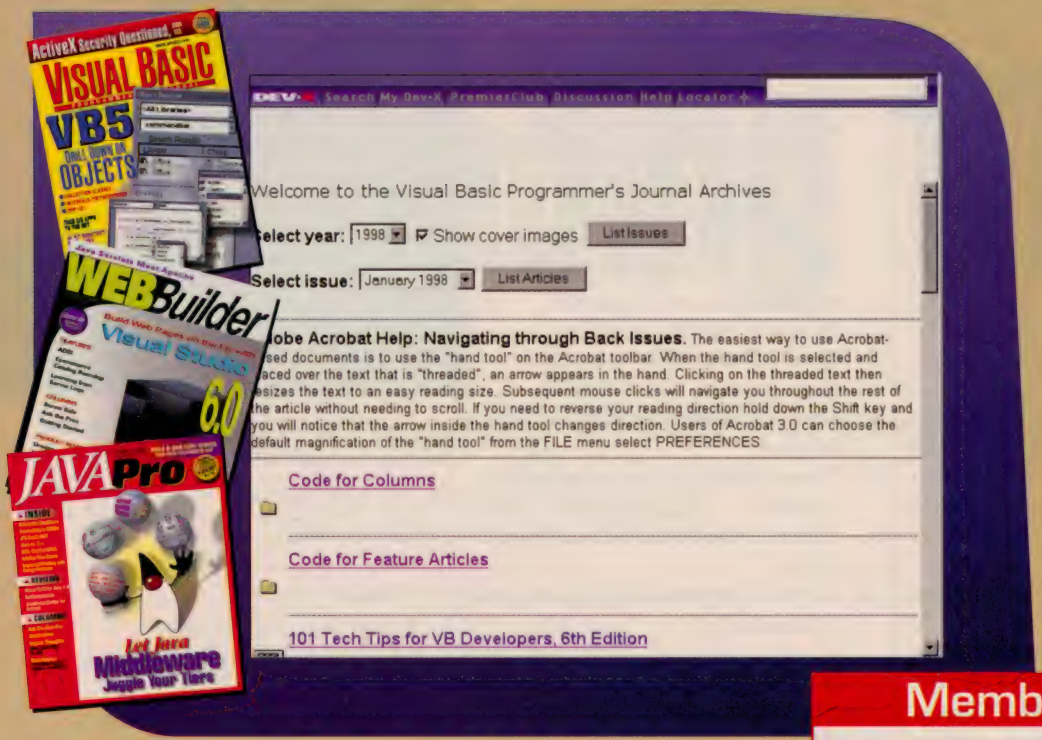
It's easy to see how major problems could arise if shared resources are not properly locked

Beginning Thread Development

To facilitate the development of multithreaded applications, Java provides the java.lang.Thread class and the java.lang.Runnable interface. Classes wishing to become multithreaded will either directly extend Thread or—if an alternate inheritance tree is needed—implement the Runnable interface. The Thread class provides a series of methods of use to developers, however, the most important is the `run()` method. Once a Thread object is given a turn to execute, all of the code in its `run()` method will be executed. Note that a Thread object's `run()` method is never to be called directly. Some internal housekeeping is performed by the virtual machine before running an thread, and the `start()` method is provided for this reason. To begin a thread's execution, instantiate a new object that extends Thread and then call its `start()` method. In Listing 1, ThreadDemo and ThreadRunner, demonstrate this.

Attention Developers

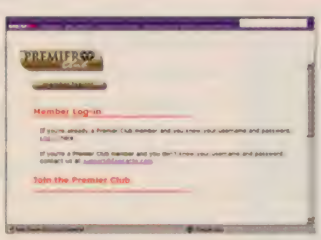
Search Articles. Download Code.



PREMIER club

Join now and Premier Club membership is ONLY $20 for six months. You'll get an additional 25% off if you already subscribe to *VBPJ*, *WEB Builder* or *Java™ Pro*.

The Development Exchange's (DevX's) Premier Club offers you access to the largest resource of programming information and tools available anywhere. Your Premier Club membership starts with immediate, unlimited access to information and downloadable code from the entire archive of FTP's leading programming titles—information that will save you tons of coding time.



And as a Premier Club member, your access extends to any of the unique development zones and sites within the DevX family of web sites—all with related articles, key tools, and dedicated programming information on VB, C++, Java, and Enterprise development.

To start your membership, point your browser to DevX's Premier Club area, and follow the instructions.

Member Benefits

- **Access to All Previous Issues.** Over 1,000 articles from *Visual Basic Programmer's Journal*, *WEB Builder*, Avatar, *Java™ Pro*, and *Getting Started with Visual Basic*.
- **Code!** Members gain admission to the entire archive area of DevX, with complete searchable text from all these publications.
- **Even More Code.** Extra apps with complete source, plus special utilities.
- **Email Product Reviews.** Get e-mail issues of Visual Programming++ (VP++), with product reviews of the latest development tools as they are released.
- **Tools and Links.** Unlimited access to DevX's searchable product database, the only comprehensive, online listing of every significant programming tool on the market.
- **Club Privileges.** Special discounts, discussions, services, and much more.

Enroll Today at

www.premier-club.com

DEV·X The Development Exchange

www.devx.com

Microsoft, Windows, Visual Basic, Visual C++ are registered trademarks of Microsoft Corporation. Java is a registered trademark of Sun Microsystems.
The Development Exchange, DevX, CareerLink, and Ask the Pros are trademarks of Fawcette Technical Publications.
Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA 94301-2500. Phone: 650.833.7100; Fax: 650.853.0230; www.devx.com.

Listing 2. The Runnable Interface. This listing shows a revised version of the example in Listing 1 using the Runnable Interface to facilitate multithreading.

```
// ThreadDemoRunnable.java

/**
 * Simple thread demo using the Runnable interface
 */
public final class ThreadDemoRunnable implements Runnable {
   private String _sName;

   public ThreadDemoRunnable(String sName) {
      _sName = sName;
   }

   public void start() {
      Thread t = new Thread(this);
      t.start();
   }

   public void run() {
      for(int i=0; i<50; i++) {
         System.out.println(_sName+": "+i);
      }
   }
}

// ThreadRunnerRunnable.java

/**
 * Starts two ThreadDemoRunnable objects
 * in motion.
 */
public final class ThreadRunnerRunnable {

   public static void main(String[] args) {
   ThreadDemoRunnable demo1 =
      new ThreadDemoRunnable("Java");
   ThreadDemoRunnable demo2 =
      new ThreadDemoRunnable("C++");
   demo1.start();
   demo2.start();
   }
}
```

Listing 3. The Power of the Multithreaded Application. Each of the applications shown here contain many objects that are constantly changing state.

```
// ColorDemo.java

import java.awt.*;

/**
 * Creates 9 ColorChanger objects and attempts to
 * set them all in motion.
 */
public class ColorDemo extends Frame {
   private ColorChanger[] changers;

   public ColorDemo() {
      changers = new ColorChanger[9];
      setLayout(new GridLayout(3,3));
      for(int i=0; i<9; i++) {
         changers[i] = new ColorChanger();
         add(changers[i]);
      }
   }

   public void startChanging() {
      for(int i=0; i<9; i++) {
         changers[i].startChanging();
      }
   }

   public static void main(String[] args) {
      ColorDemo demo = new ColorDemo();
      demo.pack();
      demo.setVisible(true);
      demo.startChanging();
   }
}

// ColorChanger.java

import java.awt.*;

/**
 * Class that draws a square on the screen and changes its
 * background color every 1000 milliseconds.
 */
public final class ColorChanger extends Canvas {
   public Color color;

   public ColorChanger() {
      color = Color.black;
   }

   /**
    * Keep chaning colors until the process is stopped
    */
   public void startChanging() {
      while(true) {
         // change the color
         color = new Color(randomNumber0To255(),
         randomNumber0To255(), randomNumber0To255());
         // repaint the screen
         repaint();
         // chill
         try{  Thread.sleep(1000); }
         catch(   InterruptedException ie ) {}
      }
   }

   /**
    * Generate a random number from 0..255. Not too accurate,
    * but for our purposes it does not really matter.
    */
   private int randomNumber0To255() {
      int iLarger = (int)Math.round(Math.random()*1000);
      int iSmaller = (int)Math.round(Math.random()*100);
      return ((iLarger <= 255) ? iLarger : iSmaller);
   }

   /**
    * Redraw the screen
    */
   public void paint(Graphics g) {
      g.setColor(color);
      g.fillRect(0,0,100,100);
   }

   /**
    * Preferred size of the component
    */
   public Dimension getPreferredSize() {
      return new Dimension(100,100);
   }
}
```

ThreadDemo, extends Thread and prints its logical name along with the numbers 0–49 to the standard out. ThreadRunner, instantiates two ThreadDemo objects and starts them running at the same time.

In the ThreadDemo class, multithreading is achieved by extending the Thread class, however due to Java's single inheritance limitation, this is not always possible. When a class must be both multithreaded and extend a class other than Thread, the Runnable interface combined with a Thread object allows for multithreading. To achieve threading using the Runnable interface, a class implements the interface, places its thread code in a run() method, and passes a reference of itself to a Thread object using its constructor. When the start() method of the newly created Thread object is called, the code in the run() method of the class that implemented the Runnable interface is executed. This version involves slightly more work, but it is an equally elegant solution. Listing 2 shows a revised version of the example in Listing 1, using the Runnable interface.

Comparing Threaded with Non-Threaded Applications

The examples in Listings 1 and 2 show how to implement multiple threads, but they don't fully demonstrate the power of making an application multithreaded. In the next example, we'll write two versions of an application in which many objects are constantly changing state and

Threading may be one of the most under-utilized features in Java

Listing 4. An Alternative Approach. Correcting the problems in Listing 3, this code allows the squares in this application to properly change colors.

```
//ColorDemoWithThreads.java

import java.awt.*;

/**
 * Creates 9 ColorChangerWithThreads objects and
 * gets them to all change colors at about the same time.
 */
public class ColorDemoWithThreads extends Frame {
   private ColorChangerWithThreads[] changers;

   public ColorDemoWithThreads() {
      changers = new ColorChangerWithThreads[9];
      setLayout(new GridLayout(3,3));
      for(int i=0; i<9; i++) {
         changers[i] = new ColorChangerWithThreads();
         add(changers[i]);
      }
   }

   public void startChanging() {
      for(int i=0; i<9; i++) {
         changers[i].startChanging();
      }
   }

   public static void main(String[] args) {
      ColorDemoWithThreads demo =
         new ColorDemoWithThreads();
      demo.pack();
      demo.setVisible(true);
      demo.startChanging();
   }
}


//ColorChangerWithThreads.java

import java.awt.*;


public final class ColorChangerWithThreads extends Canvas {
   public Color color;

   public ColorChangerWithThreads() {
      color = Color.black;
   }

   /**
    * Invoked to indicate that it's time to start changing
    * colors. Starts a new ColorSwitcher object in motion.
    */
   public void startChanging() {
      ColorSwitcher switcher = new ColorSwitcher();
      switcher.start();
   }

   /**
    * Inner class used to change the current color in a
    * unique thread
    */
   class ColorSwitcher extends Thread {
      /**
       * Keep chaning colors until the process is stopped
       */
      public final void run() {
         while(true) {
            // change the color
            color = new Color(randomNumber0To255(),
               randomNumber0To255(), randomNumber0To255());
            // repaint the screen
            repaint();
            // chill
            try{  Thread.sleep(1000); }
            catch(   InterruptedException ie ) {}
         }
      }

      /**
       * Generate a random number from 0..255. Not too accurate,
       * but for our purposes it does not really matter.
       */
      private int randomNumber0To255() {
         int iLarger = (int)Math.round(Math.random()*1000);
         int iSmaller = (int)Math.round(Math.random()*100);
         return ((iLarger <= 255) ? iLarger : iSmaller);
      }

   } // end inner class

   /**
    * Redraw the screen
    */
   public void paint(Graphics g) {
      g.setColor(color);
      g.fillRect(0,0,100,100);
   }

   /**
    * Preferred size of the component
    */
   public Dimension getPreferredSize() {
      return new Dimension(100,100);
   }
}
```

those changes are reflected onscreen. A class called `ColorChanger` extends java.awt.Canvas, allocates a 100-pixel square for itself, and changes its color every 1,000 milliseconds. An additional class, `ColorDemo`, extends Frame, adds nine `ColorChanger` objects to its screen, and puts them in motion. The code for these classes is contained in Listing 3.

If you type the code from the two classes and start the application, a rather unfortunate situation occurs. Figure 1 shows the application running, but only the top-left square changes color. Because the control flow is passed to the first `ColorChanger` object, the object enters an infinite loop and none of the other `ColorChanger` objects are given a chance to begin changing.

Listing 4 contains a revised version of the application where all squares are able to properly change color. While examining the code in Listing 4, pay attention to the alternative approach used to achieve threading in the `ColorChangerWithThreads` class. Because `ColorChangerWithThreads` must extend Canvas, it cannot logically extend Thread as well. Instead of implementing the Runnable interface, the JDK 1.1 inner class feature is utilized. Inner classes allow a new class to be declared and used, all within the namespace of the outer class. The inner class can only be instantiated by its parent class, and it can call methods present both in itself and in its parent. Our inner class, `ColorSwitcher`, directly extends Thread and in its `run()` method generates random color values, and then repaints the parent class. As shown in Figure 2, the new version of the application achieves exactly what we were hoping for.

Now that we've covered threading and implemented two demonstration applications, stop for a minute,pat yourself on the back, and open a bottle of bubbly. You've covered some of the most difficult aspects of the Java threading model and, assuming that you understand everything, you're 90% of the way towards being able to put multithreading on your rèsumè. In this final example, we'll implement an application that demonstrates the pitfalls of not locking access to shared resources.

Locking Access to Shared Resources

In our earlier discussion, we examined what could happen if two threads tried to withdraw funds from the same bank account at the same time, demonstrating the problems that can occur if access to shared resources is not locked. Listing 5 contains code for three classes that demonstrate this potential snafu. The Account class holds access to an account balance and provides methods to make deposits and withdrawals. The `WithdrawMaker` class is passed to an Account object and makes withdrawals as long as funds continue to exist. The third class, AccountSNAFU, instantiates a series of transactions and waits to see which one causes a problem first. Since the allocation of resources to threads is not always dependable, an exaggerated number of simultaneous transactions are started.

Banks don't like their customers to be overdrawn, so we'll rewrite the `WithdrawMaker` class so that access to the Account object is locked before the balance check is performed and unlocked after the withdrawal is made. Shared resources are locked in Java using the synchronized keyword. This keyword is used to either mark a code block or an entire method, during which certain resources are locked. If a method is marked as synchronized—for example, `public synchronized withdraw()`—it will execute in its entirety before control is given to another object. If the synchronized modifier is used to lock access to a block of code, that modifier will also specify the resource that is to be locked and access to that resource by anyone else will be prevented. Listing 6 contains an updated version of the `WithdrawMaker` class, now titled `Safe-`

JAVA TRAINING
THAT AMAZES.
"If you enjoy learning using interactive media, I recommend the Java™ programming series from MindQ." says Miko Matsumura, Java Evangelist, JavaSoft.
MindQ
A Knowledge Universe Company
For more information call 800.646.3008 or visit www.mindq.com

Listing 5. Locking Shared Resources. This code demonstrates the possible problems that can result when shared resources are not properly locked.

```
// Account.java

/**
 * Class used to represent a unique bank account
 */
public final class Account {
   private float _fBalance;

   public Account() {
      _fBalance = 1000f;
   }

   public float balance() {
      return _fBalance;
   }

   public void deposit(float fAmount) {
      _fBalance += fAmount;
   }

   public void withdraw(float fAmount) {
      _fBalance -= fAmount;
   }
}


//WithdrawMaker.java

/**
 * Class that makes account transactions, and sometimes
 * causes an account to become overdrawn.
 */
public class WithdrawMaker extends Thread {
   public Account _account;

   public WithdrawMaker(Account account) {
      _account = account;
   }

   public void run() {
      // while funds exist, widthraw 75 units of money
      boolean bContinute = true;
      while(bContinute) {
         bContinute = _account.balance() >= 75f;
         if(bContinute) _account.withdraw(75f);
      }

      if(_account.balance() < 0) {
         System.out.println("SNAFU!: "+_account.balance());
      }
   }

}

// AccountSNAFU.java

/**
 * Class that starts a lot of transactions going, usually
 * causing the account in question to become overdrawn.
 */
public class AccountSNAFU {

   public static void main(String[] args) {
      Account account = new Account();
      WithdrawMaker[] wm = new WithdrawMaker[10000];
      for(int i=0; i<10000; i++) {
         wm[i] = new WithdrawMaker(account);
      }

      for(int i=0; i<10000; i++) {
         wm[i].start();
      }
   }
}
```

WithdrawMaker. Here, access to the Account object is locked before a transaction begins and unlocked after it completes.

At this point you should stop again, bypass the back pat, and finish off that bottle of bubbly. Although there other aspects of threads to explore, such as starting and stopping threads, managing threads as a group, and setting thread priorities, we've covered the basics of the Java threading model. If you incorporate them into your work over the next week you should have little trouble becoming a thread expert in no time flat. Feel free to send champagne to the author as well. ■

Listing 6. A Safe Withdrawal. This code updates the WithdrawMaker class. Now access to the Account object is locked before the balance check is performed and unlocked after the withdrawal is made.

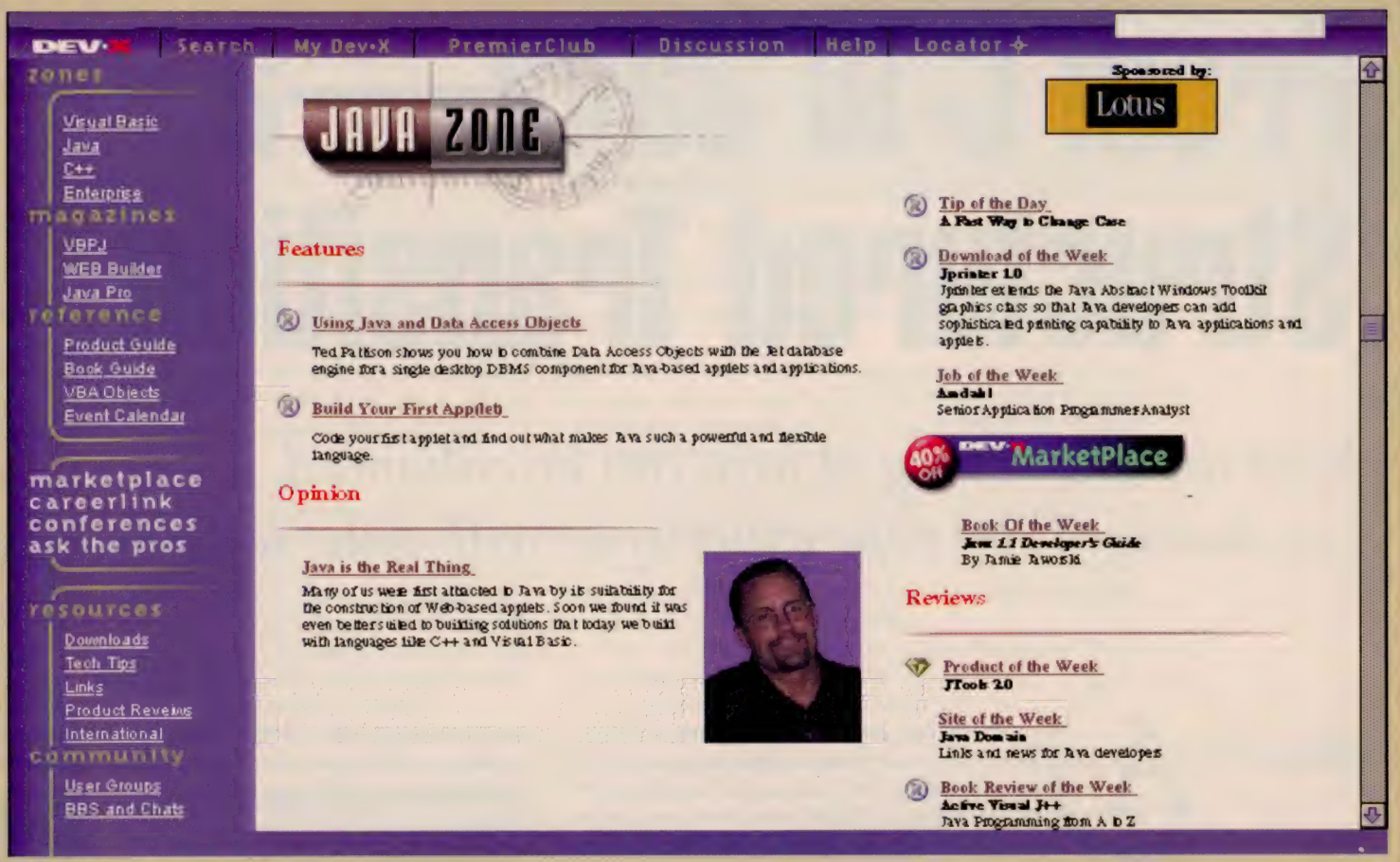
```
// SafeWithdrawMaker.java

/**
 * Class that safely makes account transactions
 */
public class SafeWithdrawMaker extends Thread {
    public Account _account;

    public SafeWithdrawMaker(Account account) {
        _account = account;
    }

    public void run() {
        // while funds exist, widthraw 75 units of money
        boolean bContinute = true;
        while(bContinute) {
            synchronized(_account) { // lock access to the account
                bContinute = _account.balance() >= 75f;
                if(bContinute) _account.withdraw(75f);
            } // unlock access to the account
        }

        if(_account.balance() < 0) {
            System.out.println("SNAFU!: "+_account.balance());
        }
    }

}
```

Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

Threading 101
Locator+ Code: JP0898LD

www.java-zone.com



100% Pure Development Solutions. Delivered Daily Online.

It's all here on DevX's Java Zone.

Jump on today and get immediate access to the most comprehensive source of expert advice, information, code, and essential resources to help you develop serious Java applications, including:

- **Technical articles and code** from *Java™Pro, Visual Basic Programmer's Journal* and *WEB Builder* magazines
- **Daily programming tips**—plus a library of thousands more
- **Download Java tools**—new applications and applets featured weekly
- **Search Products**—from our database of over 5,000 programming tools
- **Product Reviews**—hundreds of reviews updated weekly
- **Forums**—dedicated to Java development
- **40% off Books of the Week**—plus editors' picks and reviews of top programming titles
- **Conference updates**—register online

And that's just the beginning.

DEV·X
The Development Exchange

Explore the rest yourself at
www.java-zone.com.

Java-Zone is a member of the family of Windows and interactive development sites located on The Development Exchange at http://www.devx.com.
Java is a registered trademark of Sun Microsystems. The Development Exchange and DevX are trademarks of Fawcette Technical Publications.
Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA 94301-2500. Phone: 650.833.7100; Fax: 650.853.0230.

From C to Java—a Structured Transition

Your understanding of Java can be enhanced (or damaged) by your experience with older languages

by Peter Varhol

As a former and possibly future computer science professor, I'm fascinated with the process of learning to program. Not the process of learning a particular language. That's an exercise in rote memorization and you can know a language without knowing how to program. I'm interested in the process of taking someone who knows the syntax and semantics of a particular language and make a programmer out of them.

For me, an academic course I taught on data structures accomplished this feat. Data structures involve the use of a language to create building blocks such as linked lists, queues, stacks, and data abstracts. Most programs use these structures to organize, store, and manipulate the data used in the program. Writing a program involves creative use of these building blocks. For this reason, I preferred to teach data structures as a fundamental step in the development of a programmer rather than a step in learning a language. Once you know data structures, you can learn any language, and program in that language.

Or so I thought. I bailed out of academia in the spring of 1996, about the time Java 1.0 was emerging from Sun. Like many programmers, I downloaded the beta JDK and got my feet wet. After pondering both the language and its implications for programming, I knew it was time to leave academia, at least for awhile. In my final data structures course, I did a ten-minute roundup of the Java object model, and concluded by saying, "If Java becomes the next mainstream programming language, everything we've discussed this semester will become obsolete."

It was a dramatic statement, and, as it turns out, not entirely true. The theory of data structures is typically taught in C or Pascal and, in those languages, they serve two primary purposes—to create basic building blocks and create fundamental algorithms for manipulating those blocks. The same ideas apply in Java. It's a large topic, and one I can't cover in a single article. However, I can provide a flavor of how the use of data structures in a traditional language like C is similar to and different from Java's approach.

Peter Varhol is a software writer, consultant, and college professor in New Hampshire. He can be reached at peterv@mv.mv.com.

Data Structures Have Different Priorities

Java is a fascinating language, reminiscent of Smalltalk but with less integration between the language and underlying environment. Most trade and how-to books on Java focus on the language syntax, how it is used, the packages available, and the mechanics of putting a program together, especially one that draws pretty windows and graphics on the screen.

As a general rule, however, several of the features that are considered interesting and important in Java have little relevance to data structures. Most programmers would consider graphics and GUI components to be vital features of any language. But in learning to program using data structures, the Abstract Windowing Toolkit (AWT) is little more than an implementation detail. You can, as Microsoft has done, replace it with another windowing system. Has changing the AWT changed the Java language? No, however if you've learned to program the AWT, but haven't learned the language, you're out of luck.

Other features, such as platform independence, which is the subject of much debate in the Java community, affects Java only insofar as its memory references are virtual rather than physical. This requires almost no understanding of pointers, which is critical with other languages. Threads, an important feature in advanced Java programming, typically aren't considered important in learning the fundamentals of programming. This may change as Java is used more as a teaching language. Likewise, applets, an innovative feature of Java and useful for Web programming, typically require user interface components. Applets are distinguishable from applications primarily by their size and function and their calling conventions, making the distinction between applets

and applications a relatively trivial detail. Were I teaching data structures in Java, I might consider providing a user interface "shell" for use in an applet to test and run programs implementing underlying data structures.

The last feature to consider is the use of pointers. Is it possible to discuss pointers, or more properly references, in Java without knowing the essential role pointers play in C? Can one be a competent programmer, even in Java, without understanding the underlying memory allocation and management issues? I think the answer is no. It's easy for the novice to imagine the underlying computer hardware as a black box over which the programmer has no control. If I write a program that doesn't run well, I can blame it on the hardware. If I understand how the underlying memory and other hardware work, however, I can use that knowledge to make my program run better. It's too easy to forget this concept in Java unless you understand how a memory reference relates to how the computer stores the data.

Types and Data Structures

Data structures are as important for programming in Java as they are in C, but for different reasons. In C, you use data structures to create a program from the ground up. Data structures form building blocks of programs by providing data in an organized form. They are used to move data into and out of functions that manipulate that data. By combining data structures with the functions that manipulate the data, we build up whole programs.

Data structures hide the details of a particular implementation of an algorithm. I may have a function that inserts data into a list, but I shouldn't have to know how that function works in order to call it. A call like `Insert(item)` should work without my being concerned about whether the method handles that particular type of item.

All of this is true in Java as well, but there's more. In object-oriented programming, data structures not only represent data but also the operations allowed on that data. Although, you can implement functions in C data structures, they don't have to be related to the encapsulated data, and they don't support inheritance.

Types are the building blocks of data structures. This is true whether it's an object in Java or a struct in C. Consider primitive types. There's not a lot of difference between Java and C. Java's eight primitive types, the byte, short, int, long, float, double, char, and boolean, are similar to those found in C. C also lets you create more complex types using struct as a building block.

Java's analogue to struct is the object. The fundamental data structure building block in Java is the object, which provides a wealth of tools and techniques for building custom structures. Data structures in an object-oriented language also promote component reuse. To do this, the object provides a shell, so to speak. You can encapsulate a linked list, for example, into an object, so the list structure also contains the methods for manipulating the list.

You don't really know a language until you understand its data structures

Pointers and Pointers

Aside from objects, the biggest difference between C and Java is the definition and use of memory pointers. In C, data can be stored by value or by reference. In practice, virtually everything that's not a primitive is stored by reference, using pointers, for the sake of flexibility and control. Consequently, the difference between value and reference becomes very clear, especially because you have to use a different notation (* and &) for accessing data by reference. To use the two different types of reference, you have to clearly understand what they are and why you should use one or the other.

In Java, storing by value and reference is seamless, because the notation is the same. The only thing that differentiates access by value and reference is the data type itself—all primitives are stored by value and all objects, including strings, arrays, and file streams by reference.

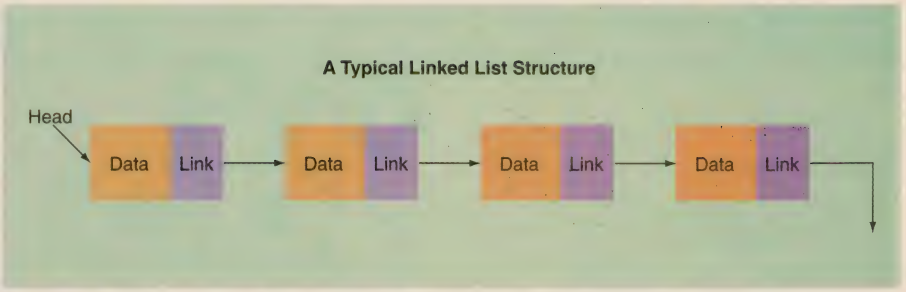


Figure 1. A Fundamental Building Block. Linked lists use pointers to dynamically connect records in a variable-length list. New records can be added to the linked list by creating a new data record and then linking the last record to the new one. Linked lists provide a basis for alternative designs for other data structures, such as stacks and queues.

Listing 1. Implementing a linked list in C. The C version is appreciably longer than the equivalent Java code. The relationship between directly manipulating pointers and hard-to-track bugs is left as an exercise for the reader—but only those who continue to use C and C++.

```
#define    NULL0
typedef    charDATA

struct LinkList
{
  DATA         d;
  struct LinkList *next;
}

typedef    struct LinkList ELEMENT;
typedef    ELEMENT    *LINK

LINK list(s);
char s[];
{
  LINK  head = NULL, tail;
  int   i;

if (s[0] != '<CCOLOR "FF0000">0')
{
  head = (LINK) malloc (sizeof (ELEMENT));
  head->d = s[0];
  tail = head;
  for (i = 1; s[i] != '<CCOLOR "FF0000">0'; ++i)
  {
    tail->next = (LINK malloc (sizeof (ELEMENT));
    tail = tail->next;
    tail->d = s[i];
  }
  tail->next = NULL;
}
return (head);
}
```

Listing 2. Objectifying Data Structures. These packages, Interface and LL, do two things. The Interface package abstracts methods that describe the state of the list. The LL package defines the list iterator, the class used for all access to the list.

```
package Interface;

public interface List
{
  boolean isEmpty();
  void makeEmpty();
}

package LL;

import Exceptions.*;

Public interface List Itr
{
  void insert(Object value) throws ItemNotFound;
  boolean find(Object value);
  void remove(Object value) throws ItemNotFound;
  boolean InList();
  Object GetItem();
  void zeroth();
  void first();
  void advance();
}
```

The primary difference between the two approaches is that it's not possible to do pointer arithmetic or other operations on Java memory addresses, as you would with C. Addresses for objects in Java are relative (or virtual) addresses arbitrarily assigned in the virtual machine. There is no reason why you might want the pointer address because you have no idea where the VM is storing objects in the underlying memory.

But using pointers in Java raises a more basic question from the standpoint of learning and using languages in general. You don't have to know the difference between value and reference in using Java; the language definition and the VM take care of it for you. In C, the concept is fundamental to programming. Whether Java tells you enough about the distinction for you to apply it in other languages is still an open question.

Is consciously making a choice between using pointers and reference structures or using non-pointer value structures an essential part of programming? Or can we simply choose the structure, and let the underlying language definition and programming system decide how that structure gets stored?

Ultimately, this is a religious argument, with no right answer. Sure, if you're programming in languages like C, C++, or Pascal, it's vital to know how to specifically store by reference. In Java, and languages like Fortran or Lisp, it's not. As long as you understand the difference between value and reference, which are important for variable scoping in method calls, how you apply them is up to the language.

Lists, Stacks, and Queues

Linked lists are one of the most fundamental data structures for building programs, and I've spent a lot of time teaching them to beginning programming students. They typically involve the use of pointers to dynamically connect records in a variable-length list. It's often diagrammed as in Figure 1.

Building a linked list typically consists of three basic operations—instantiating an object of the appropriate type, reading data into the object, and joining the item to the list. Use a simple class structure for the List object:

```
class ListNode
{
  Object data;
  ListNode link;
}
```

You can then implement the basic operations on the linked list:

```
last.link = new ListNode();
last = last.link;
last.data = value;
last.link = null;
```

In C, you begin by creating a similar data structure using struct and typedef. (See Listing 1.)

You can demonstrate that a linked list is possible in Java, but that doesn't mean that it's useful. Would you use a linked list in a language whose user interface and communications strengths make its use of such structures a minor consideration? Sure you would. However, you still have to create the data and underlying operations that make the program work.

The linked list provides a basis for alternative designs for other data structures, such as stacks and queues. A stack, a last-in, first-out (LIFO) storage device, can be constructed as a linked list where `add()` and `remove()` methods (called "push" and "pop" in stack vernacular) are permitted access only to the last item in the list. Likewise, a queue is a first-in, first-out (FIFO) structure that can be a linked list where the `enqueue()` method operates only on the last item in the list, and `dequeue()` only on the first item in the list. For building these and other data structures, I may use structs in C and objects in Java, but a programmer who learned one approach will easily identify and be able to work with the other.

The other primary alternative for building stacks and queues is the array. Comparing linked lists to array implementations for higher level structures is useful because it illustrates some of the fundamental differences in programming approaches. It's worthwhile providing at least two implementations for data structures to illustrate the concept of tradeoffs—there is no single way to program for all applications.

These and more complex structures can be created in both Java and C, and are readily comparable. Both Java and C have arrays as a defined part of the language, and they are used in the same way.

Abstracting Data Structures

What we'd like to do in an object-oriented system, like Java, is abstract the details of the data structures and include the behavior as a part of the object. Objects, fortunately, do this naturally. For example, the packages in Listing 2 do two things. The Interface package abstracts methods that describe the state of the list. The LL package defines the list iterator, the class used for all access to the list.

Objects highlight some of the primary differences between Java and C. On the surface an object looks like a record and can be used to create structures in similar ways. Underneath, an object offers more flexibility for defining properties and methods, and abstracting those characteristics.

But data structures do more to tie together different languages, because the building blocks are largely the same; it's just the implementations that are different. Understanding linked lists, queues, and stacks, and other data structures enable you to create programs of moderate complexity in any language.

I remain ever more convinced that you don't really know a language until you understand its data structures. You may know its syntax, but not how it behaves in real use. My walk down this comparison path has produced one practical result—I've asked to get back into the classroom to teach a course on data structures with Java. ■

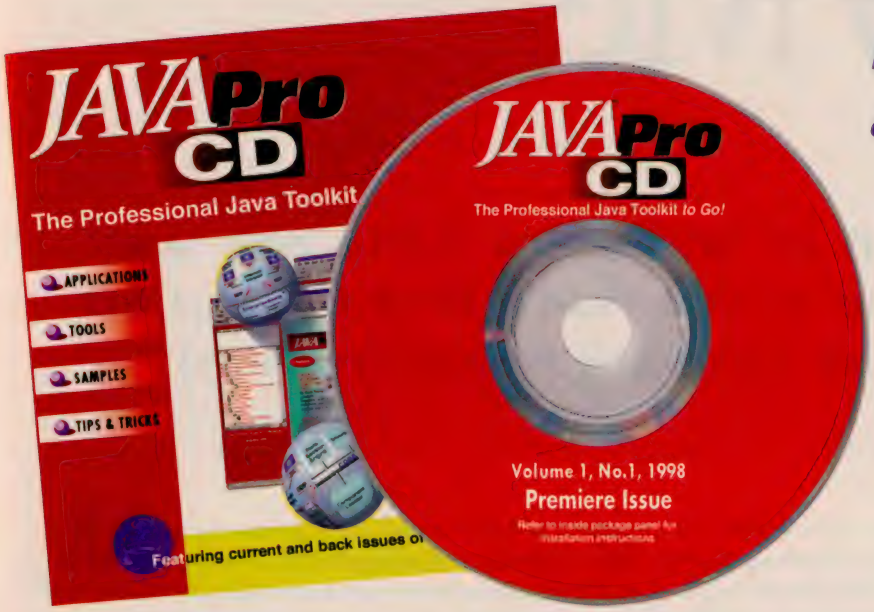
Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

From C to Java—a Structured Transition
Locator+ Code: JP0898PV

Professional Java Toolkit to Go:

JAVAPro CD



It's like getting an entire Web site on a CD – without the world wide wait!

6 Java™ Pro Issues only $99.95

All subscriptions get FREE Access to the Development Exchange Premier Club

A $40 value at no extra cost. You'll get augmented code, shareware, new product demos and other utilities. Access DevX at www.devx.com

Chock full of valuable applications, tools and samples you can actually load onto your machine, run, and use to develop serious Java applications, including:

- Java development tools and utilities
- The latest Java runtime environment
- Key beta software and trial versions of new products
- Lots of Java Doc for those times when you can't be online
- Tips, tricks and techniques to help you design and build effective Java applications
- Current and back issues of *Java Pro*
- A wide-ranging directory of Java-related products
- In-depth product information and white papers from key Java tools and middleware vendors
- Informative chapters from the best new books on Java
- Documentation of key additions to the Java standard

Includes the trial version of Borland JBuilder™!

All tools and demos are hand-picked by the Editors of *Java Pro* to complement the magazine.
All presented in an approachable, engaging and–most importantly–useful format. What would normally take hours to download will be available at your fingertips in an instant.

Order today! CALL 800.848.5523
from outside the U.S. call 650.833.7100 • fax 650.321.3818

or order online at www.devx.com

FTP FAWCETTE TECHNICAL PUBLICATIONS
209 Hamilton Avenue • Palo Alto, CA 94301-2500

Please allow 4 weeks for delivery of first issue.

YES! I'd like to order the *Java Pro* CD. Call 800.848.5523 or fill out this coupon and fax it to 650.853.0230.

❍ Subscription: $99.95 for one year (6 CD issues)

Name _____

Address _____

City _____ Zip _____ Country _____

Phone [ ] _____ Fax [ ] _____

Email _____

Payment enclosed ❍ Charge my: ❍ Visa ❍ Master Card ❍ AMEX

Card # and expiration date _____

Signature _____

International orders please add $15 for shipping and handling. JP0898

Fawcette Technical Publications • 209 Hamilton Avenue • Palo Alto, CA 94301 • 650-833-7100 • www.devx.com

Java is a trademark of Sun Microsystems. DevX and The Development Exchange are trademarks of Fawcette Technical Publications. JBuilder is a trademark of Borland, International, Inc.

Socket to Me

Although much of the Java APIs are well designed, some areas need fine tuning to facilitate robust system programming

by Daniel F. Savarese

Daniel Savarese holds a B.S. in Astronomy and an M.S. in Computer Science, both from the University of Maryland College Park. Daniel is the author of the OROMatcher regular expression library for Java and currently directs Java product development at ORO Inc. Reach him at dfs@oroinc.com.

Recently I ran into one of the more annoying design flaws in the Java core libraries. I had previously implemented a set of networking classes, all of which subclass a general socket client class that takes care of the common tasks of connecting, disconnecting, and maintaining references to input and output streams. Then a developer told me he'd like to make all the socket client subclasses use SSL sockets. Naively, I suggested he use the `Socket.setSocketImplFactory()` static method to change the underlying socket implementation for his application. As it turns out, `setSocketImplFactory()` is not helpful at all.

You can only call `setSocketImplFactory()` once in your entire application and, when called, it changes the underlying SocketImpl type used for all sockets created in your program. In addition, virtually no third party libraries provide SocketImpl implementations because to do so requires native code. It's easier to provide a pure Java implementation of a customization, like SSL, by subclassing Socket and wrapping its input and output streams. Given that all third party socket customizations are implemented by subclassing Socket, I added support for the developer's request by creating a SocketFactory interface defining a set of methods that instantiated Sockets. This turned out to be quite a boon. Now if a developer needs to add proxy support to one of my network classes or request special browser privileges before opening a socket connection, all he has to do is provide his own SocketFactory implementation and bind it to the client by using a setSocketFactory method. It is a clean solution that avoids messing with SocketImpl and allows different Socket implementations to be used throughout a single application.

Just when I thought I was done, another developer pointed out a problem that has bothered me since I started programming in Java. Every time you create a Socket, it connects to the specified host. If the connection attempt fails, it throws an IOException. If the connection attempt succeeds, you wind up with a newly created Socket. What if you don't want to wait an unspecified amount of time for a connection to be established? Let's say you want to wait 10 seconds to establish a connection and fail if no connection is established after those 10 seconds. There is no way to do this in pure Java because of the way the Socket and SocketImpl classes are designed.

The way things stand today, you have to wait indefinitely until a Socket constructor returns or throws an exception in order to establish a connection. This is unacceptable for a wide range of network client applications that require a certain degree of responsiveness and fault-tolerance. At first I thought there must be a workaround. The SocketImpl abstract class defines two protected `connect()` methods and the PlainSocketImpl class has a public `setOption()` method that can be used to set a timeout for a connection attempt. But PlainSocketImpl is a hidden class local to the `java.net.*` package. There is simply no way to access the SocketImpl or PlainSocketImpl functionality without writing your own native code. This problem could be solved by making the no-argument Socket constructor public (it is currently declared protected) and adding a set of `connect()` methods to Socket. This would allow programmers to create a Socket, set a timeout using `setSoTimeout()`, and then attempt a connection, rather than being forced to wait on the system default time for a connection to timeout. Although much of the Java APIs are well designed, there are some areas that need fine tuning to facilitate robust system programming. The Socket and SocketImpl classes constitute one such area.

Q Handling Unix Signals

Is there a way in Java to handle Unix signals (i.e., to trap a Control C signal)?

Listing 1. Printing a Component. The tricky part of printing a Java Component is realizing that you have to invoke getPrintJob() through the default Toolkit instance.

```
import java.awt.*;
import java.awt.event.*;

import com.sun.java.swing.*;


public final class Print {

   public static final void main(String[] args) {
      WindowListener exitListener;
      final JFrame frame = new JFrame();
      final JButton button = new JButton("Print");

      exitListener = new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            Window window;
            window = e.getWindow();
            window.setVisible(false);
            window.dispose();
            System.exit(0);
         }
      };

      button.addActionListener(new ActionListener() {

         public void actionPerformed(ActionEvent e) {
   if(e.getSource() == button) {
      PrintJob job;
      Graphics g;

      job = Toolkit.getDefaultToolkit().getPrintJob(
         frame, "A Print Job",
                  null);
      if(job == null) {
         JOptionPane pane;
         JOptionPane.showMessageDialog(
            frame, "Could not get print job!","Error",
            JOptionPane.ERROR_MESSAGE);
         return;
      }

      g = job.getGraphics();
      frame.print(g);
      job.end();
   }
      }
      });

      frame.getContentPane().add(button);
      frame.addWindowListener(exitListener);
      frame.pack();
      frame.show();
   }
```

A You can handle Unix signals only by writing native code. However, the Java virtual machine uses various signals for its own purposes, and providing your own signal handlers will usually break the virtual machine. It is highly recommended that you don't install your own signal handlers.

Q Java Meets NT

How can I run a Java server application as an NT service?

A You can run a Java server application as an NT service if you use the Microsoft SDK for Java.(Download the latest version from Microsoft's Web site at http://www.microsoft.com/java/sdk/.) The SDK includes a package called com.ms.service which contains a Service class for creating NT services. You also need to turn the Java application into an executable NT service using the `jntsvc.exe` utility provided with the SDK. If you look in the bin directory of the SDK, there is a jntsvc subdirectory that contains `jntsvc.exe` and an example Java program implementing an NT service. More information is available in the Microsoft SDK for Java documentation.

Q New Print Classes

How do I print a component?

A Java 1.1 introduced the `java.awt.PrintJob` and `java.awt.PrintGraphics` classes. It also added the print(Graphics) and printAll(Graphics) methods to `java.awt.Component`, and the printComponents(Graphics) method to `java.awt.Container`. To print a component, you only need to concern yourself with the `PrintJob` class and the print(Graphics) method in Component.

To print a component, create a PrintJob instance, obtain its drawing context through `getGraphics()`, use that as an argument to print(Graphics), and end the print job by calling the `PrintJob end()` method. The only tricky part of this procedure creating the PrintJob instance. You don't use the new operator, rather, get the default Toolkit instance, using `Toolkit.getDefaultToolkit()`, and create a PrintJob through `getPrintJob`(Frame, String, Properties). The Toolkit class mediates between the pure Java environment and the system dependent facilities.

Below is an example of a button that prints itself. Note that you should call `print()`, and not `printComponents()`. Also, on most systems, `getPrintJob()` will present the user with a print configuration dialog, and block until one user dismisses the dialog. If the user dismisses the dialog by pressing the cancel button, `getPrintJob()` will return null, so you should check the return value before using it (see Listing 1).

Q Getting File Dates

Is there any way to get a file's date with Java?

A Unfortunately, there doesn't seem to be a platform-independent way to do this. The `lastModified()` method in `java.io.File` returns a system dependent value that can only be used to compare the dates of files, and then only the time of last modification, not the time of creation. However, the Unix and Windows95 versions of the Sun JDK represent this value as the number of seconds since 00:00 January 1, 1970. Knowing this, you can convert the value to a date using the GregorianCalendar class. But if you do, you have to keep in mind that there is no guarantee the Sun JDK will continue to represent the time of last modification in this manner. There is also no guarantee that any other JVM will use the same representation format. In other words, you can't use this value in a portable manner.

Latest Issue–Summer '98

COMPLETE CD-ROM TOOLKIT TO

Build A Web Site Today!

$29.95 for 4 Quarterly Issues

The *Web Builder* CD-ROM features the latest version of more than 55 tools, over 150 complete articles and code from *Java™Pro* and *Visual Basic Programmer's Journal*, plus software, links, basics, and guidance to continue and expand learning.

TOPICS COVERED

- Planning and Design
- Authoring HTML
- Client Programming
- Server Programming
- Site Management

SOFTWARE

Working demos and trial versions including:

- Server Software
- ActiveX and Java Utilities
- Security Software
- Image and Audio Editors
- Customized IE 4.0 Browser

Plus, FREE Web hosting setup and one month's service by Interland, Inc.!

Subscribe to the *Web Builder* CD-ROM today at the low rate of $29.95 for 4 quarterly issues—you'll save 25% off the newsstand price! Canada/Mexico, add $6/year. All other countries please add $15 for shipping/handling.

CALL TODAY TO SUBSCRIBE!

1.800.848.5523
or 650.833.7100 (outside the U.S.)

Visual Basic is a registered trademark of Microsoft Corporation. ActiveX is a trademark of Microsoft Corporation. Java is a trademark of Sun Microsystems.

FTP Fawcette Technical Publications | 209 Hamilton Ave. • Palo Alto, CA 94301-2500
Tel: 650.833.7100 • Fax: 650.853.0230

or order online at

http://www.devx.com

Ask the Java Pro

Q Appending File Data

Is there a way to open a file for writing in append mode?

A The `FileOutputStream` and `FileWriter` classes both have constructors with the following signature (String, boolean). The String parameter is a filename, and the boolean parameter indicates whether or not the file should be opened in append mode. Set the boolean parameter to true if you want to append to a file. Alternatively, you can open a RandomAccessFile and seek to the end of the file before writing to it.

Q Using Date or Calendar

I'm writing a class that reads a log file containing timestamps. My StringTokenizer returns the timestamp as a string and I then have to store it in a date container. Should I use Date or GregorianCalendar? Also, how do I use DateFormat to parse the timestamp?

A The first question is difficult to answer. Normally I would say use GregorianCalendar because so much of the Date class is deprecated. However, when you parse the timestamp String using DateFormat, you'll end up with a Date object. The basic rule is if you need to modify or add and subtract your times, you should use a Calendar instance. If you need a read-only date representation that can be compared to other dates, the Date class is usually sufficient, but you probably should still stick to a Calendar instance.

To parse the timestamp, use the `parse()` method in DateFormat. First create a DateFormat that uses the default locale and style. You can do this by calling the `getDateInstance()` static method in DateFormat. Now simply invoke the `parse()` method in the resulting DateFormat instance. The `parse()` method returns a Date instance. You can convert a Date to a Calendar with the Calendar `setTime()` method. `parse()` will understand most common date representation formats, however, if your log timestamps are in a format it doesn't understand, you may have to write your own parser.

Q Support for Tips

How do I get a tool tip to appear in a Java application?

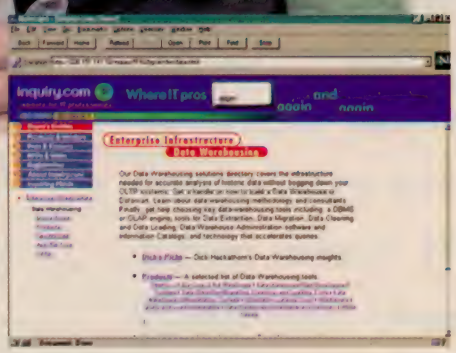
A Tool tips were not a part of the AWT packages in the 1.0.2 and 1.1 releases of Java. To use tool tips in a 1.0.2, you must either create your own custom tool tip code or use a third party package. For a 1.1 application, the best thing to do is use the Swing classes. You can download the Swing classes as part of the Java Foundation Classes from JavaSoft's Web site (http://java.sun.com/products/jfc/index.html). Swing and the JFC are a standard part of the AWT packages in Java 1.2.

Many of the Swing components have built-in tool tip support. You can use them to call `setToolTipText(String)`, defined in JComponent. For components that do not have built-in tool tip support, you may create a `JToolTip` instance, set its text with `setTipText(String)`, and bind it to a component with `setComponent(JComponent)`. To create your own custom tool tips, you may override the `createToolTip()` method in JComponent. ■

Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

Socket to Me
Locator+ Code: JP0898AP



"We found the IT answers we needed at inquiry.com."

When you need answers on development tools and techniques, turn to www.inquiry.com.

Our editors and "Ask the Pro" professionals provide unique insights on real-world development problems.

Now that inquiry.com is part of The Development Exchange suite of Web sites, you also have access to the largest database of independent product reviews, technical tips, and our Buyers Guide with information on over 5,000 tools for professional developers.

Nothing matches the efficiency and scope of www.inquiry.com.

Get the IT answers you need, free, today.

Find us at www.inquiry.com

PC MAGAZINE MAY 97 THE 100 TOP WEB SITES

inquiry.com
answers for IT professionals

© Inquiry.com, the inquiry.com logo, and The Development Exchange are trademarks of Fawcette Technical Publications.

Observing Your Windows

The Observer design pattern helps you keep data objects and display objects separated

by James W. Cooper

James W. Cooper is a computer science researcher and the author of a number of books in the field. He is currently working on his thirteenth book, on Java Design Patterns, for Addison-Wesley-Longman.

Now that Java has matured as a language, we find ourselves writing more and more complex programs, with multiple windows, buttons, toolbars, and all the decorations and widgets of a powerful user interface. Windows has raised our expectations for these interfaces over the years and we've come to expect programs that are elegant in appearance as well as sophisticated in construction.

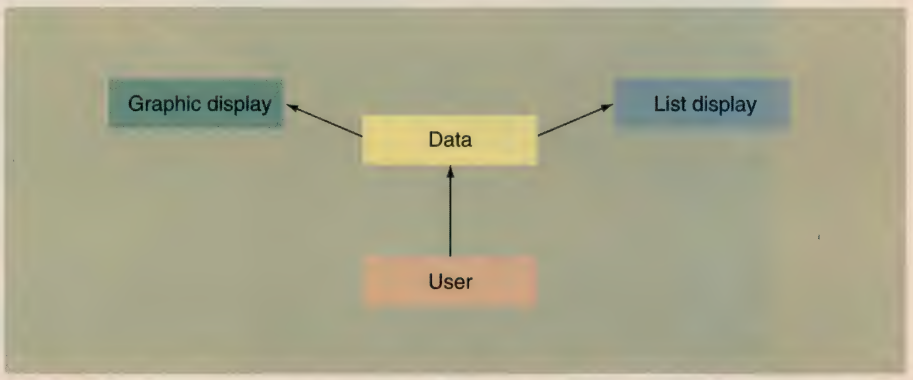


Figure 1. The Observer Design Pattern. A data object, observed by two different display objects. The user can change the data and see the effect in both displays.

The Java Foundation Classes (JFC), which were called the "Swing" classes during development, are a replacement for, and an enhancement of, the original AWT classes. You'll find them in the Swing package for Java 1.1 and included in Java 1.2. They're also supported by the Java Plug-In for your Web browser, so you don't have to ask users to download a bunch of new class files.

These new classes are lightweight and should provide better graphical performance than the original AWT classes. They're also considerably more sophisticated in the functions they provide. Further, they provide a pluggable look and feel, which lets you make your program look like Windows on Windows systems and like Motif on Unix systems. A Macintosh look and feel is also in the works. To acquaint you with these powerful classes, we'll use them in this and following columns, whenever we need a user interface example. You'll also discover a number of design patterns inherent in these classes, which make them easier to use than the original AWT classes. In this column, I'll show you a sample of the Observer pattern and how it is used in the JFC.

The Observer Pattern

In our new, more sophisticated windowing world, we often find that we'd like to display data in more than one form at a time, and have all of the displays reflect changes we make in that data. For example, you might represent stock price changes as a graph and as a table or list box. Each time the price changes, we'd expect both representations to change without any action on our part.

We expect this sort of behavior because it is common in a number of Windows applications, like Excel. There is nothing inherent in Windows, however, to support this activity, and programming this behavior directly in Windows in C or C++ is complicated. In Java, however, we can easily make use of the Observer design pattern to cause our program to reflect changes made in the data.

The Observer pattern assumes that the object containing the data is separate from the objects that display the data, and that these display objects observe changes in that data. This is simple to grasp and simple to illustrate, as we see in Figure 1.

When we implement the Observer pattern, we usually refer to the data as the Subject and each of the displays as Observers. Each of these observers registers its interest in the data by calling a public method in the Subject. Then, each observer has a known interface that the subject calls when the data changes. We can define these interfaces as follows:

```
abstract interface Observer {
// notify the Observers
// a change has taken place
   public void sendNotify(
      String s);
}
//=======================
abstract interface Subject {
// tell the Subject you are
// interested in changes
   public void registerInterest(Observer obs);
}
```

The advantage of defining these abstract interfaces is that you can write any sort of class objects you want as long as they implement these interfaces, and you can declare these objects to be of type Subject and Observer no matter what else they do.

Watching Colors Change

Let's write a simple program to illustrate how we can use this powerful concept. Our program shows a display frame containing three radio buttons named Red, Blue, and Green, as shown in Figure 2.

This main window is the Subject or data repository object. We create this window using the JFC classes in this simple code:

```
public class Watch2L extends JFrame
   implements ActionListener,
ItemListener, Subject {
   Button Close;
   JRadioButton red, green, blue;
   Vector observers;
// ------------------------
   public Watch2L()    {
   super("Change 2 other frames");
// list of observing frames
   observers = new Vector();
// add panel to content pane
   JPanel p = new JPanel(true);
   p.setLayout(new BorderLayout());
   getContentPane().add(
        "Center", p);

// vertical box layout
   Box box = new Box(BoxLayout.Y_AXIS);
   p.add("Center", box);
// add 3 radio buttons
   box.add(red = new JRadioButton("Red"));
   box.add(green = new
JRadioButton("Green"));
   box.add(blue = new
 JRadioButton("Blue"));

// listen for button clicks
   blue.addItemListener(this);
   red.addItemListener(this);
   green.addItemListener(this);

// all part of the same group
ButtonGroup bgr = New ButtonGroup();
   bgr.add(red);
   bgr.add(green);
   bgr.add(blue);
```

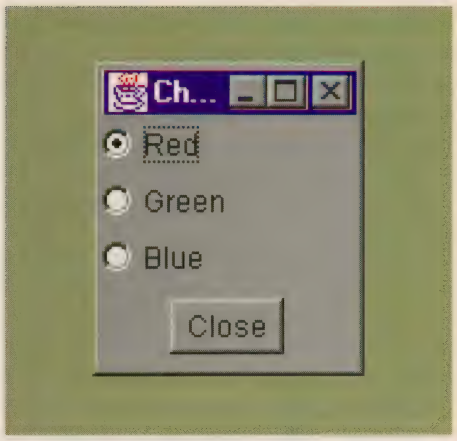


Figure 2. Collecting Input. Users make their selection using these radio buttons. Their choice becomes the subject data. Check boxes or a listbox could be also be used as the subject.

Note that our main frame class implements the Subject interface. This means it must provide a public method for registering interest in the data in this class. This method is the `registerInterest()` method, which just adds Observer objects to a Vector:

```
public void registerInterest
            (Observer obs)    {
//add to list in Vector
      observers.addElement(obs);
}
```

Now we create two observers, one, which displays the color and its name, and another, which adds the current color to a list box:

```
//------create observers--------
   ColorFrame cframe =
        new ColorFrame(this);
   ListFrame lframe =
        new ListFrame(this);
```

When we create our `ColorFrame` window, we register our interest in the data in the main program:

```
class ColorFrame extends Jframe
  implements Observer {
   Color color;
   String color_name = "black";
   JPanel p = new JPanel(true);
//-------------------
   public ColorFrame(Subject s)    {
    super("Colors");
// set frame caption
    getContentPane().add(
          "Center",  p);
    s.registerInterest(this);
// register with Subject
    setBounds(100, 100, 100, 100);
    setVisible(true);
   }
//--------------------
public void sendNotify(
            String s)    {
```

Another problem for observers may arise when the data can change in several ways

```
// Observer notified of change here
      color_name = s;
// save color name and
// set background to that color
      if(s.toUpperCase().equals("RED"))
         color = Color.red;
      if(s.toUpperCase().equals("BLUE"))
        color =Color.blue;
      if(s.toUpperCase().equals("GREEN"))
        color = Color.green;
      setBackground(color);
}
//---------------------------
   public void paint(Graphics g)    {
   g.drawString(color_name, 20, 50);
   }
```

Meanwhile, in our main program, every time someone clicks on one of the radio buttons, it calls the `sendNotify()` method of each Observer who has registered interest in these changes by simply running through the objects in the observers Vector:

The Observer pattern assumes that the object containing the data is separate from any display objects

```
   public void
itemStateChanged(ItemEvent e) {
   // responds to radio button clicks
   // if the button is selected
      if(e.getStateChange() ==
      ItemEvent.SELECTED)
notifyObservers((JRadioButton)
                e.getSource());
   }
   //-----------------------------
   private void notifyObservers(
              JRadioButton rad)    {
// sends text of selected
// button to all observers
   String color = rad.getText();
   for (int i = 0;
        i <
observers.size(); i++) {
         ((Observer)

(observers.elementAt(i))).

sendNotify(color);
      }
   }
```

In the case of the `ColorFrame` observer, the `sendNotify()` method changes the background color and the text string in the frame panel. In the case of the `ListFrame` observer, it adds the name of the new color to the list box. We see the final program running in Figure 3.

Figure 3. Two Observers, One Subject. Both observers receive the same data from the subject, but each responds differently.

The Message to the Media

What kind of notification should a subject send to its observers? In this carefully circumscribed example, the notification message is the string representing the color itself. When we click on one of the radio buttons, we can get the caption for that button and send it to the observers. This assumes that all the observers can handle that string representation. In more realistic situations, this might not always be the case, especially if the observers could also be used to observe other data objects. Here we undertake two simple data conversions: We get the label from the radio button and send it to the observers, and we convert the label to an actual color in the `ColorFrame` observer. In more complicated systems, we might have observers that demand specific, but different, kinds of data. Rather than have each observer convert the message to the right data type, we might use an intermediate class to perform this conversion. These classes are called Adapter classes, and we'll look at how we use them in a future column.

Another problem for observers may arise when the data of the central subject class can change in several ways. We could delete points from a list of data, edit their values, or change the scale of the data we are viewing. In these cases we either need to send different change messages to the observers or send a single message and then have the observer ask what sort of change has occurred.

The JList as an Observer

What about the list box in our color changing example? We saved it for last in this article because the JList is different in concept from the List object in the AWT. Although you can display a fixed list of data in the JList by putting the data into a Vector or String array, if you want to display a list of data that might grow or otherwise change, you need to put that data into a special data object derived from the `AbstractListModel` class, and then use that class in the constructor to the JList class. Our `ListFrame` class looks like this:

```
class ListFrame extends JFrame
   implements Observer {
   JList list;
   JPanel p;
   JScrollPane lsp;
   JListData listData;

   public ListFrame(Subject s)    {
    super("Color List");
    // put panel in the frame
    p = new JPanel(true);
    getContentPane().add(
            "Center",  p);
    p.setLayout(new BorderLayout());
    // Tell the Subject we are
    // interested
    s.registerInterest(this);

    //Create the list
      // the list model
    listData = new JListData();
     // the visual list
    list = new JList(listData);
    // the scroller
    lsp = new JScrollPane();
    lsp.getViewport().add(list);
    p.add("Center", lsp);
    lsp.setPreferredSize(
        new Dimension(100,100));
```

```
    setBounds(250, 100, 100, 100);
    setVisible(true);
  }
  //-----------------------
  public void sendNotify(
       String s) {
    listData.addElement(s);
  }
}
```

We name our `ListModel` class `JListData`. It holds the Vector that contains the growing list of color names:

```
class JListData extends AbstractListModel
{
  private Vector data;
// the color name list
  public JListData()    {
    data = new Vector();
  }
  public int getSize()    {
    return data.size();
  }
  public Object getElementAt(
    int index) {
    return
        data.elementAt(index);
  }
// add string to list and
// tell the list about it
  public void addElement(
        String s)    {
    data.addElement(s);
    fireIntervalAdded(
        this, data.size()-1,
        data.size());
  }
}
```

Whenever the `ColorList` class is notified that the color has changed, it calls the `addElement()` method of the `JListData` class. This method adds the string to the Vector and then calls the `fireIntervalAdded()` method. This base method of the `AbstractListModel` class connects to the JList class, telling that class that the data has changed. The JList class then redisplays the data as needed. There are also equivalent methods for two other changes; `fireIntervalRemoved()` and `fireContentsChanged()`. These represent the three kinds of changes that can occur in a list box. Here each sends its own message to the JList display.

The MVC Architecture as an Observer

What does this all sound like? An Observer pattern! This is a practical example from the JFC classes of how to make fairly sophisticated use of the Observer pattern. The JList is the Observer and the `JListData` class is the subject. You'll see by reading the Swing documentation that the Combo box has similar properties.

In fact, all of the visual components derived from JComponent can have this same division of labor between the data and the visual representation. In JFC parlance, this is referred to as the Model View Controller (MVC) architecture, where the data are represented by the Model, and View, by the visual component. The Controller is the communication between the Model and View objects, and it may be a separate class or it may be inherent in either the model or the view. This is the case for the JFC components, and they are all examples of the Observer pattern we've just discussed.

As we dig deeper into how we use Java to build elegant programs we'll see that these sort of design patterns continue to play an important role in our programming. ■

Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

Observing Your Windows
Locator+ Code: JP0898JT

TechTIP

Printing a Report

Printing a simple report from Java isn't as simple as you might expect. You use the Toolkit class to create a `PrintJob`, but the method requires a Properties object, which appears to have no subsequent value. Each line of text has to be positioned graphically, requiring you to decide on a font and obtain some details of the character sizes:

```
Properties prop = new Properties();
Toolkit tkit = Toolkit.getDefaultToolkit();
PrintJob pjob = tkit.getPrintJob(yourFrame, "Print ...", prop);
Font mono = new Font("Monospaced", Font.PLAIN, 12);
FontMetrics metrics = tkit.getFontMetrics(mono);
int linespace = metrics.getHeight();
int descender = metrics.getDescent();
Graphics page = null;
int pagesize = 0;
int location = 0;
```

The `PrintJob` will provide a Graphics object for each page of the report, but you need to keep track of the position of each text line so you can determine when it's time to eject a page and start another. It's essential to set the Graphics object's Font, because there is no default. You can't reliably obtain the paper size until after the Graphics object is created, because it may trigger display of a platform dependent dialog box, prompting the user for a choice of printer, paper, and orientation. The loop, which prints lines of your report, should contain code with the following sequence:

```
location = location + linespace;
if (location > pagesize) {
  if (page != null) page.dispose();
  page = pjob.getGraphics();
  page.setFont(mono);
  pagesize = pjob.getPageDimension().height;
  location = linespace;
}
page.drawString(yourtext, 0, location - descender);
```

After the loop you need to eject the last page and terminate the PrintJob:

```
if (page != null) page.dispose();
pjob.end();
```

Thankfully, the units of the `FontMetrics`, `PageDimension` and String locator are consistent and are typically 1/72nd of an inch. The Java SDK refers to them as pixels. Comparing the width and height returned by the `getPageDimension` method enables you to determine paper orientation, but if you're interested in the actual paper size, you're out of luck. The `getPageResolution` method is supposed to provide the pixels-per-inch, but in most implementations these are not the same pixels! Despite SDK documentation to the contrary, the value returned seems to be the printer's dots-per-inch.

–Tom Yarker

Visual J++ 6.0 and ActiveX Data Objects

With Microsoft's latest release of Visual J++, building Windows applications that use the Common Object Model has gotten easier

by Andy Wilson

When I received the Visual J++ 6.0 preview, I walked (very quickly) to my workstation and installed my new toy. I rarely get so excited about development environments, but I had heard that there were many improvements over Visual J++ 1.1. During three days of exploring this new spin on Java, I made a few discoveries.

First, Visual J++ 6.0 is great for developing GUI applications. From day one, I have found layout managers to be the most annoying area of Java. I never cared for the idea of a dynamic layout scheme where my controls varied in location, spacing, and size depending on the size of my window. However, Visual J++ 6.0 has a Visual Basic-like GUI editor in which you can drag-and-drop controls on a Microsoft Form object. Within minutes, I was able to drag-and-drop a UI that would have taken me a couple hours to write using traditional layout managers.

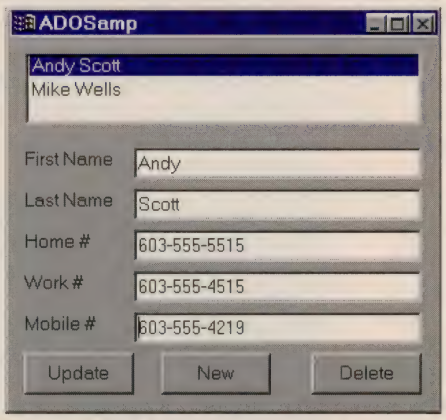


Figure 1. Yet Another Phonebook Application. Building simple database applications using ADO is easy with the new release of Visual J++, and much of the code is automatically generated.

Second, almost everything is now based on the Component Object Model (COM). Many of the interesting new extensions to the Visual J++ flavor of Java are COM-based. More importantly, Visual J++ 6.0 produces COM objects that work within C++ and Visual Basic applications. It didn't take more than 20 minutes to write my first Java COM object. Without a doubt, Visual J++ is the simplest COM development environment I've ever worked with.

Third, the Windows Foundation Classes (WFC) contain a set of classes that expose the bulk of Windows to a Java application. The WFC is almost perfectly impure. It takes full advantage of the operating system at the cost of platform independence, because using the WFC locks your application to the Windows operating system.

Fourth, database applications are easier to write using ActiveX Data Objects (ADO). Visual J++ includes a set of wrapper Java classes that expose the ADO interfaces. These classes come in two layers. The first layer contains the raw ADO COM interfaces that give you enough functionality to create an instance of an ADO object. However, these interfaces require you to write a fair amount of code to initialize the object. The second layer simplifies the task of creating an ADO object by performing these initializations automatically.

These new technologies really aren't surprising. What amazed me was that I could build a small database application within half an hour. When I'd worked with the JDBC in the past, I had to create all the code that bound my `java.sql.ResultSet` object to various controls on my application. Even on a good day, creating the simplest database application using the JDBC took a few hours.

To get a taste for the new features of Visual J++ 6.0, let's look at how quickly you can build a simple database application.

Database Development

Figure 1 shows the UI for our sample phone book application, which works like most others you've seen. The application queries the database for a set of records and displays the

Andy is a software developer at Compuware-NuMega Labs. Reach him at AndyW@Numega.com.

Listing 1. **ADO Sample.** The ADOSamp application is your typical phone book. However, note that most of the code is in the initForm() method. This code is automatically generated by Visual J++ when adding elements to the user interface.

```
// ADOSamp.java

import wfc.app.*;
import wfc.core.*;
import wfc.ui.*;
import wfc.data.*;

public class ADOSamp extends Form
{
  // Data Members
  private Recordset m_recordset = null;
  private int lastRecord = 0;
  private boolean newRecord = false;

  // Event handlers

  /**
   * ConnectionComplete:
   *Called by the DataSource after attempting to
   *connect to the database
   */
  private void ConnectComplete(Object sender,
    wfc.data.ConnectionEvent e)
  {
    if ( e.adStatus != AdoEnums.EventStatus.OK )
    {
      // the new message box object
      MessageBox.show( "Error connecting to database",
          e.toString() );
      return;
    }

    //get the Recordset from the DataSource
    m_recordset = dataSourceNumbers.getRecordset();

    // fills the listbox
    buildList();
  }

  /**
   * buttonDelete_click:
   * Called in response to a user clicking on the Delete button.
   * Deletes the selected record from the Recordset
   */
  private void buttonDelete_click(Object sender, Event e)
  {
    if ( newRecord == false )
    {
      m_recordset.delete();
      m_recordset.requery();
      buildList();
    }
  }
  /**
   * buttonNew_click:
   * Called in response to the user clicking on the New button.
   * Generates a new record within the Recordset.
   */
  private void buttonNew_click(Object sender, Event e)
  {

    if ( newRecord == false )
    {
      m_recordset.addNew();
      newRecord = true;
    }
  }

  /**
   * buttonUpdate_click:
   * Called in response to the user clicking
   * on the Update button.
   * Saves the Recordset's changes to the database. Then
   * performs a requery operation.
   */
  private void buttonUpdate_click(Object sender, Event e)
  {
    m_recordset.update();
    m_recordset.requery();
    buildList();
  }

  /**
   * listBoxRecord_change:
   * Called when the user selects an item from the listbox.
   * Updates the Recordset's cursor location
   */
  private void listBoxRecord_change(Object sender, Event e)
  {
    if ( newRecord == true )
    {
      // If the user clicks on the New button
      // and then selects a record from the list,
      // then abort the new record.
      m_recordset.cancelUpdate();
      newRecord = false;
    }

    // Move relative to the previous selected record
    // within the Recordset.
    m_recordset.move( listBoxRecords.getSelectedIndex() -
              lastRecord );

    lastRecord = listBoxRecords.getSelectedIndex();
  }

  // Regular class methods
  /**
   * buildList:
   * Fills a listbox with some information
   * from the Recordset.
   */
  private void buildList()
  {
    listBoxRecords.removeAll();
    // Move to the first record in the Recordset.
    m_recordset.moveFirst();

    // Loop until we hit the end of the Recordset.
    while ( m_recordset.getEOF() == false )
    {
      // Field information is a two step process.
      // First, get the field by either column
      // name or index.
      // Second, retrieve the data from the
      // resulting Field object.
      listBoxRecords.addItem(
        m_recordset.getField( "FirstName" ).getString() +
        " " + m_recordset.getField( "LastName"
        ).getString() );
      m_recordset.moveNext();
    }

    m_recordset.moveFirst();
    listBoxRecords.setSelectedIndex( 0 );
    lastRecord = listBoxRecords.getSelectedIndex();
  }

  public ADOSamp()
  {
    initForm();
  }

  public static void main(String args[])
  {
    Application.run(new ADOSamp());
  }

  // The following code was the result of adding controls
  // to the application's form.
  /**
  * NOTE: The following code is required by the Visual J++
  * form designer.  It can be modified using the form
  * editor.  Do not modify it using the code editor.
  */
```

Listing continued on next page.

www.enterprise-zone.com

Need Help Tackling the Enterprise?

Find all the information you need on DevX's enterprise-zone—

the most comprehensive independent source of technical information for Windows and interactive developers.

Join today and get immediate access to powerful tools, valuable code and DevX's vast resource of development information to help you produce advanced division-level applications, including:

- **Technical articles and code** from *Visual Basic Programmer's Journal*, *WEB Builder*, and *Java™ Pro* magazines
- **More Code**—download the best shareware and product demos
- **Programming tips**—plus a library of thousands more
- **Products**—find the right tool from a database of over 5,000 tools
- **Reviews of 100's of products**—with new reviews posted weekly
- **Dedicated Development forums**
- **Conference Updates**—register online for VBITS and the NT Enterprise Developer's Conference
- **Buy Programming Books Online** Save up to 40% on selected titles, plus get editors' picks and reviews of top programming titles

Enterprise-Zone is a member of the family of Windows and interactive development sites located on The Development Exchange at http://www.devx.com

DEV·X The Development Exchange

Windows and Visual Basic are registered trademarks of Microsoft Corporation. The Development Exchange and DevX are trademarks of Fawcette Technical Publications.
Fawcette Technical Publications, 209 Hamilton Avenue, Palo Alto, CA 94301-2500. Phone: 650.833.7100; Fax: 650.853.0230; www.devx.com

results in various fields. The Listbox displays all the names of people within the database, while the edit boxes display individual fields. The surprises come not from the interface, but from how easy it is to build.

In Visual J++ 6.0, you develop an interface with a series of drag-and-drop operations from a toolbox. The toolbox contains a set of controls that expose a set of properties. The properties of an object manipulate its attributes, such as color, size, position, and text. Additionally, the control may also expose a set of events. Control properties and events are set via the Properties window. The `initForm()` method contains the resulting code from all of these drag-and-drop operations.

A control can be anything from a simple label to a Windows common control. However, one important distinction between a Java UI object and a WFC UI object is that the WFC object exposes a `wndProc()` method. The `wndProc()` method is the Windows message process for the control. Overriding this method allows the application to read messages from the operating system. Therefore, it's no longer necessary to hook a window to get to its message process, unless you want to read messages for another window.

Other non-UI controls provide additional functionality, such as database access. One such control is the `wfc.data.ui.DataSource`. The DataSource control allows a Java application to attach to a database. The DataSource object creates a set of ADO objects and these ADO objects interact with the database.

ADO

ADO is broken down into a hierarchy of objects. The most important class is the `wfc.data.Connection` class, which provides the functionality to connect to a database. From the Connection class we get a variety of methods that allow the application to perform get and set operations on locks, cursor types, and cursor locations. The Connection class also allows the application to perform queries and updates.

The `wfc.data.Recordset` class objects are the results of queries. The Recordset object exposes three additional objects. The first is the Field object that represents a single field for a given row. The application uses the Field object to retrieve the actual data. The second object Recordset exposes is the `wfc.data.Fields` object, a collection of fields for the current row. The application can use the Fields object to access a single Field object. The third object is the `wfc.data.Command` object. Although the Recordset uses the Command object, the Command object must first be bound to the Recordset. The Command object allows an application to build parameterized SQL commands. Command objects can simplify building and manipulating SQL operations dramatically. The DataSource object interacts with all of these objects in one way or another, and manipulating the DataSource causes indirect changes in the resulting ADO objects.

Now let's move on to the sample and see Visual J++ 6.0 in action.

The Source

The first half of the `ADOSamp.java` file (see Listing 1) contains the event handlers for various operations, like clicking on a button or selecting an entry from the Listbox. The interesting, but difficult to read, code is within the

Listing continued from previous page.

```
Container components = new Container();
ToolTip tipSamp = new ToolTip(components);
Label labelLastName = new Label();
Label labelFirstName = new Label();
Label labelHome = new Label();
Label labelWork = new Label();
Label labelMobile = new Label();
Edit editFirstName = new Edit();
Edit editLastName = new Edit();
Edit editHome = new Edit();
Edit editWork = new Edit();
Edit editMobile = new Edit();
ListBox listBoxRecords = new ListBox();
Button buttonUpdate = new Button();
Button buttonNew = new Button();
Button buttonDelete = new Button();
wfc.data.ui.DataSource dataSourceNumbers =
    new wfc.data.ui.DataSource(components);
wfc.data.ui.DataBinder dataBinderNumbers =
    new wfc.data.ui.DataBinder(components);

private void initForm()
{

  this.setBackColor(Color.CONTROL);
  this.setLocation(new Point(0, 0));
  this.setSize(new Point(319, 302));
  this.setTabIndex(-1);
  this.setTabStop(true);
  this.setText("ADOSamp");
  this.setAutoScaleBaseSize(16);
  this.setClientSize(new Point(311, 270));
  tipSamp.setRegions(new ToolTipRegion[] {});
  /* @designTimeOnly tipSamp.setLocation(
    new Point(230, 0));  */
  labelLastName.setLocation(new Point(10, 110));
  labelLastName.setSize(new Point(80, 20));
  labelLastName.setText("Last Name");
  labelLastName.setTabIndex(3);
  labelFirstName.setLocation(new Point(10, 80));
  labelFirstName.setSize(new Point(80, 20));
  labelFirstName.setText("First Name");
  labelFirstName.setTabIndex(4);
  labelHome.setLocation(new Point(10, 140));
  labelHome.setSize(new Point(80, 20));
  labelHome.setText("Home #");
  labelHome.setTabIndex(2);
  labelWork.setLocation(new Point(10, 170));
  labelWork.setSize(new Point(80, 20));
  labelWork.setText("Work #");
  labelWork.setTabIndex(1);
  labelMobile.setLocation(new Point(10, 200));
  labelMobile.setSize(new Point(80, 20));
  labelMobile.setText("Mobile #");
  labelMobile.setTabIndex(0);
  editFirstName.setBackColor(Color.WINDOW);
  editFirstName.setCursor(Cursor.IBEAM);
  editFirstName.setLocation(new Point(90, 80));
  editFirstName.setSize(new Point(210, 23));
  editFirstName.setTabIndex(9);
  editFirstName.setTabStop(true);
  editFirstName.setText("Andy");
  editFirstName.setMaxLength(50);
  editLastName.setBackColor(Color.WINDOW);
  editLastName.setCursor(Cursor.IBEAM);
  editLastName.setLocation(new Point(90, 110));
  editLastName.setSize(new Point(210, 23));
  editLastName.setTabIndex(8);
  editLastName.setTabStop(true);
  editLastName.setText("Scott");
  editLastName.setMaxLength(50);
  editHome.setBackColor(Color.WINDOW);
  editHome.setCursor(Cursor.IBEAM);
  editHome.setLocation(new Point(90, 140));
  editHome.setSize(new Point(210, 23));
  editHome.setTabIndex(7);
  editHome.setTabStop(true);
  editHome.setText("603-555-5515");
  editHome.setMaxLength(30);
  editWork.setBackColor(Color.WINDOW);
  editWork.setCursor(Cursor.IBEAM);
  editWork.setLocation(new Point(90, 170));
  editWork.setSize(new Point(210, 23));
  editWork.setTabIndex(6);
  editWork.setTabStop(true);
  editWork.setText("603-555-4515");
  editWork.setMaxLength(30);
  editMobile.setBackColor(Color.WINDOW);
  editMobile.setCursor(Cursor.IBEAM);
  editMobile.setLocation(new Point(90, 200));
  editMobile.setSize(new Point(210, 23));
  editMobile.setTabIndex(5);
```

Listing continued on next page.

Listing continued from previous page.

```
    editMobile.setTabStop(true);
    editMobile.setText("");
    editMobile.setMaxLength(30);
    listBoxRecords.setLocation(new Point(10, 10));
    listBoxRecords.setSize(new Point(290, 54));
    listBoxRecords.setTabIndex(10);
    listBoxRecords.setTabStop(true);
    listBoxRecords.setIntegralHeight(true);
    listBoxRecords.setItemHeight(16);
    listBoxRecords.setItems(new Object[] {});
    listBoxRecords.addOnSelectedItemChanged(
    new EventHandler(this.listBoxRecord_change));
    buttonUpdate.setLocation(new Point(10, 230));
    buttonUpdate.setSize(new Point(80, 30));
    buttonUpdate.setTabIndex(14);
    buttonUpdate.setTabStop(true);
    buttonUpdate.setText("Update");
    buttonUpdate.addOnClick(
      new EventHandler(this.buttonUpdate_click));
    buttonNew.setLocation(new Point(110, 230));
    buttonNew.setSize(new Point(80, 30));
    buttonNew.setTabIndex(12);
    buttonNew.setTabStop(true);
    buttonNew.setText("New");
    buttonNew.addOnClick(new EventHandler(
      this.buttonNew_click));
    buttonDelete.setLocation(new Point(220, 230));
    buttonDelete.setSize(new Point(80, 30));
    buttonDelete.setTabIndex(11);
    buttonDelete.setTabStop(true);
    buttonDelete.setText("Delete");
    buttonDelete.addOnClick(
      new EventHandler(this.buttonDelete_click));
    dataSourceNumbers.setConnectionString(
      "Provider=MSDASQL.1; Data Source=ADOSample;
      Mode=Read|Write|ReadWrite");
    dataSourceNumbers.setCommandText(
      "select * fromNumbers");
    dataSourceNumbers.setCursorType(
      wfc.data.AdoEnums.CursorType.STATIC);
    dataSourceNumbers.setLockType(
      wfc.data.AdoEnums.LockType.OPTIMISTIC);
    dataSourceNumbers.setSort(null);
    dataSourceNumbers.setUserId(null);
    dataSourceNumbers.setPassword(null);
    dataSourceNumbers.addOnConnectComplete(
      new ConnectionEventHandler(this.ConnectComplete));
    /*
    @designTimeOnly dataSourceNumbers.addOnLocation(
      new Point(130, 20));
    */
    dataBinderNumbers.setDataSource(dataSourceNumbers);
    dataBinderNumbers.setDataMember("");
    dataBinderNumbers.setBindings(
      new wfc.data.ui.DataBinding[] {
      new wfc.data.ui.DataBinding(
        editFirstName, "Text", "FirstName"),
      new wfc.data.ui.DataBinding(
        editHome, "Text", "HomePhone"),
      new wfc.data.ui.DataBinding(
        editLastName, "Text", "LastName"),
      new wfc.data.ui.DataBinding(
        editMobile, "Text", "MobilePhone"),
      new wfc.data.ui.DataBinding(
        editWork, "Text", "WorkPhone")});
    /*
    @designTimeOnly dataBinderNumbers.setLocation(
      new Point(130, 40));
    */
    this.setNewControls(new Control[] {
      buttonDelete,
      buttonNew,
      buttonUpdate,
      editMobile,
      editWork,
      editHome,
      editLastName,
      editFirstName,
      labelMobile,
      labelWork,
      labelHome,
      labelLastName,
      labelFirstName,
      listBoxRecords});
    dataSourceNumbers.begin();
  }
  // NOTE: End of form designer support code

  public static class ClassInfo extends Form.ClassInfo
  {
    // TODO: Add your property and event infos here
  }
}
```

`initForm()` method. We'll pay special attention to the dataSourceNumbers object, which is the DataSource object for the application.

The first interesting line is the `dataSourceNumbers.setConnectionString()` method. This method specifies the driver and system data source information necessary to initially connect to the database. The `setCommandString()` specifies the SQL string that we will use to perform our first query.

The next two lines are particularly important. The first line, `setCursorType()` allows us to specify the cursor type for the Connection and resulting Recordset. The cursor type, among other things, allows us to specify the ways we want to iterate through the Recordset. In a `java.sql.Resultset`, the cursor can only move in one direction, forward. Recordset objects can move in any direction and jump from one record to another depending on which cursor the application sets.

The second line, `setLockType()`, specifies the record locking mechanism. This is a critical line of code. The settings impact when locking occurs and at what level. For example, I spent a great deal of time debugging the application when I found that I only had read access to the database. It turned out that I had accidentally set the lock type to "unspecified", which is the same as read-only.

Also, note that both lines use static types from the `wfc.data.AdoEnums` class. This class contains all the static values for ADO. It helps to read through the values for this class since it defines the bulk of the parameters for many ADO methods.

A little further down, we see a call to `addOnConnectComplete()`. This method is one of several ADO event handler methods. In this case, I wanted to know if the application successfully connected to the database, before allowing the user to do anything.

Finally, I want to talk about how to bind data to the various controls. The `wfc.data.ui.DataBinder` class binds a Recordset's fields to properties within a control. For example, to bind string data to an edit box, I would bind the field's string data to the text property of the edit box. Whenever the user moves within the Recordset, the DataBinder will automatically update the fields' properties.

Conclusion

Visual J++ 6.0 gives you a wide, new assortment of functionality, and we've just barely scratched the surface of the product in this column. Using Visual J++, you can pound out fast, efficient applications in a very short time, with much of the code generated for you. However, Visual J++ does not force you to write just system dependant applications. There is a project properties switch that disables the Microsoft extensions, allowing you to develop more traditional Java applications. Either way, Visual J++ 6.0 is a slick new Java development environment that is sure catch on. ■

Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

Visual J++ 6.0 and ActiveX Data Objects
Locator+ Code: JP0898IT

The Conference Defining Strategic Business Solutions on Windows NT

San Francisco, Oct. 28-31, 1998

EnterpriseDevelopment

Join us for four days and 50 sessions.

- Customizing ERP and Customer Service Applications
- Business Intelligence Applications for NT
- Tools, Middleware and Management
- Web-Enabling the Enterprise
- Special Business Logic Workshop
- Special Manager's Day and E-Commerce Track

Who should attend:

Systems architects, project managers, IT managers, corporate developers—anyone faced with creating large-scale applications, customizing ERP, evaluating architectures, or assembling middleware, servers, or intranets to solve enterprise-wide business challenges.

Register by September 16 to receive your Early Bird discount!

CALL TODAY!

1.800.848.5523

FOR THE LATEST INFORMATION AND TO REGISTER:

http://www.enterprisedev.com

Enterprise Development is a trademark of Fawcette Technical Publications. Windows and Windows NT are registered trademarks of Microsoft Corporation.

FTP Fawcette Technical Publications
209 Hamilton Ave.
Palo Alto, CA 94301
Phone: 650-833-7100

ViewSource

Handling Events in JavaScript

Writing code for multiple browsers requires knowledge of both JavaScript and the underlying object models

by Gary Cornell

Gary Cornell is a top-selling programming author and a noted expert on Visual Basic, Java, and Delphi. Sample chapters from his books *Core Java, Vol. 1* and *Vol. 2* are featured on the first edition of the *Java Pro CD*. A professional programmer and professor at the University of Connecticut, Gary holds a Ph.D. from Brown University. Reach him at gcornell@ix.netcom.com.

As I mentioned in my first column (View Source, "JavaScript History Lesson", *Java Pro,* February/March 1998), JavaScript and Java inherit their basic syntax from the C programming language—but this makes them distant cousins at best. One of the smaller ironies of life is that although the syntax of JavaScript and Visual Basic are worlds apart, the style used for programming them is often similar. This is because most JavaScript code, like most VB code, is *event driven*. Both JavaScript and Visual Basic programs spend most of their time waiting for an event to trigger code. You've seen some of this programming style in earlier View Source columns. For example, when you want a button click to start a process, you can write a function that will be called by the `onClick` event. In this code, the function named `handleThisButton` will be called:

```
<INPUT TYPE ="Button" VALUE="Click me"
  NAME = "TheButton"
onClick=handleThisButtonClick(this.form)">
```

Note that because of the `this.form` parameter, JavaScript will pass the routine to the form that the button is a part of. Unless and until the user clicks on the button, the code in the `handleThisButtonClick` function will lie inert.

In this column we'll go through the basic event-handling mechanism common to most versions of JavaScript. I'll call it "classic" JavaScript. This is kind of code you'll use to ensure the highest level of compatibility across browsers. Navigator 4 and Internet Explorer 4 introduced a new flavor of event handling that I'll touch on briefly at the end of this column. Unfortunately, though similar at first glance, the event handling is sufficiently different in IE4 and Navigator 4 to require you to write significantly different code to take full advantage of browser specific features.

Another consideration is that it is quite possible that the next generation of browsers will have a unified object and event-handling model. The World Wide Web Consortium (W3) has advanced a "Document Object Model" that would do this. Microsoft's model in IE4 is actually closer to the consortium's specification than Netscape's current model.

Types of Events

Generally speaking in classic JavaScript events can be triggered by one of several events:

- The browser doing something (loading or unloading a page)
- The user doing something (clicking a button, moving the mouse over a link, or moving the focus off a form ele-ment)
- A specific amount of time has elapsed from a starting point

Table 1 provides the names of the events, what causes them, and the JavaScript objects to which they apply. As the table indicates, there are a number of events and objects you might use. Happily, your script coding doesn't have to be complex. The general form of an event handler in JavaScript looks like this:

```
<HTML TAG ATTRIBUTES eventHandler="JavaScript Code">
```

The part I called "JavaScript code" is usually a function call, but for the simplest event handlers you will occasionally see inline code. You'll need to be aware of this style of coding when maintaining an existing page.

For example, here are two ways to display a page that contains "advertising". When a user moves the mouse over the www.java-pro.com link, the advertising shows in the status bar. The function approach is the first one. Although it is longer, I think it leads to cleaner code:

Listing 1. HTML Page. The Window.Event object in this code prevents the user from entering anything other than a digit in the text box.

```
<HTML>
<HEAD>

  <TITLE>Handling the onKeypress event</Title>
  <SCRIPT LANGUAGE = JavaScript>
<!--
function keyPress()
{ if ((window.event.keyCode < 48)||
      (window.event.keyCode > 57))
   {   alert ("Bad key");
       window.event.keyCode = 0;
   }
}
-->
</SCRIPT>
</HEAD>
<BODY>
<H1>
This page will warn you if you enter anything
else but a digit in the text box below.
</H1>
<BR><BR>
<INPUT ID = theText TYPE=Text onKeyPress=
  "keyPress()">
</BODY>
</HTML>
```

```
<HTML>
<HEAD>
<TITLE>Showing off the MouseOver event</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function updateStatusBar()
{ window.status ="An amazingly useful link!";
  return true;
}
</SCRIPT>
</HEAD>
<BODY>
<!-- Insert HTML here -->
<A HREF=http://www.java-pro.com/
   onMouseOver="updateStatusBar()">
  www.java-pro.com
</A>
</BODY>
</HTML>
```

In the inline version, note that whenever you use it, you need to use single quotes for the string inside the double quotes that give the JavaScript code:

```
<BODY>
<!-- Insert HTML here -->
<A HREF=http://www.java-pro.com/
   onMouseOver="window.status='An
amazingly useful link!'; return true">
www.java-pro.com
</A>
```

Event handling in Navigator 4 and Internet Explorer 4 is intimately connected to dynamic HTML

Although neither version of the event handling code is very complicated, both have the line that says "return true," which bears watching. In classic JavaScript you can cancel various actions by returning false as the value of their event handler. For example, a rogue coder could change the "return true" to "return false" in the above handler and you could click on the link forever but never be able to use it. Similarly, if you return false from an `onClick` event, it is as if the user never clicked on the button. You saw this feature at work in my previous column (View Source, "JavaScript History Lesson", *Java Pro*, June/July 1998), when I used it in the `onSubmit` event for form validation. I return true only if the form data was valid, which tells the browser to submit the data.

Another thing to bear in mind is that while many JavaScript links may have event handlers, anchors do not. For complete

Event Name	Description	Objects Affected
onAbort	Triggered when the user stops the loading of an image	image
onBlur	Triggered when the user moves the focus away from an object	frame, select, text, textarea, window
onChange	Triggered when the value (usually the text) of the object changes	select, text, textarea
onClick	Triggered when the user clicks on an item	button, checkbox, radio, link, reset, submit
onError	Triggered when a document or image can't load correctly	image, window
onFocus	Triggered when an item gains focus (opposite of onBlur)	frame, select, text, textarea, window
onLoad	Triggered when the page or an image finishes loading	image, window
onMouseOut	Triggered when the mouse pointer moves off an item	area, link
onMouseOver	Triggered when the mouse pointer moves over an item	area, link
onReset	Triggered when a form is reset	form
onSelect	Triggered when the user selects text in a text field or text area	text, textarea
OnSubmit	Triggered when the submit button on a form is pressed	form
OnUnload	Opposite of onLoad. Triggered when the document is unloaded	window

Table 1. A Simpler Set of Events. Available events in classic JavaScript let you control only certain objects.

compatibility with the largest class of browsers I suggest that whenever multiple links point to anchors in the same document, you should handle the event at the *link*, not at the anchor.

Triggering Events Via Code

Beginning with Netscape Navigator 3 you don't have to have the user trigger an event, you can do it with code yourself. For example, you might not want to require the user click on the Submit button to submit a completed form. This line triggers the click event in a button I'll call `theButton`:

```
document.theForm.theButton.click()
```

Following is a list of the events you can trigger in classic JavaScript. They consist of the event without the "on" prefix, `but the lower case is important` in Navigator 3. In Navigator 3 `document.theForm.theButton.Click()` won't work:

- blur()
- click()
- focus()
- reset()
- select()
- submit()

Life Cycle Events

The `onLoad` and `onUnload` event handlers are the life cycle events for a browser page. The load event is triggered when a page finishes loading and the unload event is triggered when the user leaves the page. The idea is that, by waiting for the `onLoad` event handler to be triggered, you can be sure that the whole page (including any images) has loaded completely:

```
<BODY onLoad="alert('Welcome to my home
page!');"
  onUnload="alert('Goodbye! Come back
again real soon now!');">
</BODY>
```

The New Event Models

Event handling in Navigator 4 and Internet Explorer 4 is intimately connected to dynamic HTML, which is a fancy way of saying that the goal is to have all items on a page be responsive to events. IE4 is much closer to this goal (and the W3 specification) than Navigator. Table 2 lists the events that both Navigator 4 and IE4 respond to. Note that the `onMouseOver` event is a bit different in the competing models.

However, because IE4 is a more complete implementation of the W3 specification, I'll spend the rest of the article using its model. The key idea in the new event model is that Table 2 should be obsolete! Every element on your Web page in IE4 (and the W3 "Document Object Model") is now responsive. For example, this code (which runs only in IE4) would pop up a message box if the user clicks on the text:

```
<HTML>
<HEAD>
<TITLE>Showing off the
MouseOver event</TITLE>
<SCRIPT
LANGUAGE="JavaScript">
function paragraphClicked()
{ alert ("You clicked on
me");
  return true;
}
</SCRIPT>
</HEAD>
```

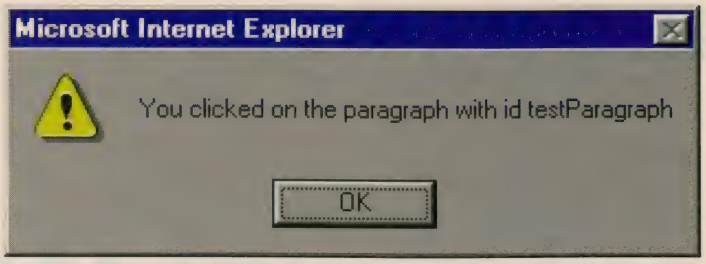


Figure 1. The srcElement at Work. Internet Explorer lets you assign events to individual paragraphs.

```
<BODY>
<!-- Insert HTML here -->
<P ID=testParagraph
onClick="paragraphClicked()"
  STYLE="font-family:Times">
  For the first time everything on the
  page is scriptable and will respond to
  events. Click me and see.>
</P>
</BODY>
</HTML>
```

The Window.Event Object

The `Window.Event` object is the key to event handling in the new W3 object model. The idea is that all the information about any event that happens inside a browser will be stored in the `Window.Event` object. In the future implementation in IE4, 21 possible pieces of information are stored in the `Window.Event` object.

For example, by using `Window.Event.SrcElement` in code you can get a reference to the object that was the source of the event. You can then use `Window.Event.SrcElement.ID` to get the name. If you can modify the function to read like this code then you'll see something like Figure 1:

```
{ alert ("You clicked on the paragraph
with id " + window.event.srcElement.id);
  return true;
}
```

Other Properties of the Window.Event Object

Table 3 gives short descriptions of the other important properties of the `Window.Event` object. The HTML page shown in Listing 1 illustrates the use of the `Window.Event` object that prevents the user from entering anything other than a digit in the text box. The key lines in the code are:

```
if ((window.event.keyCode < 48)||
      (window.event.keyCode > 57))
```

These lines check if the key has ASCII code < 48 (i.e. below "0" in ASCII order) or above a 57 (the ASCII code for a "9"). If it is, then this line of code cancels the keystroke:

```
window.event.keyCode = 0;
```

Events	Objects Affected
abort	image
blur	window, text, textarea, password, select
change	text, textarea, select
click link	area, button, radio, checkbox, reset, submit
dblclick	link
error	window, image
focus	window, text, textarea, password, select
keydown	text, textarea, password
keypress	text, textarea, password
keyup	text, textarea, password
load	window, image
mousedown	link, button, radio, checkbox, reset, submit
mouseout	link, area
mouseover	link, area
mouseup	link, button, radio, checkbox, reset, submit
move	window
reset	form
resize	window
select	text, textarea, password
submit	form
unload	window

Table 2. Overlapping Events. The common events in Internet Explorer 4 and Navigator 4.

Mouse Events

To demonstrate the mouse events, I'll show you a program that reports the current position of the mouse as you move it around the page and display the information in a text area. The secret to this example is to write code in the document_onMouseMove event that uses the window.event.x and window.event.y properties:

```
<HTML>
<HEAD>
  <TITLE>Handling the onMousemove event</
Title>
  <SCRIPT LANGUAGE = JavaScript>
  <!--

   function document.onmousemove()
   {   var temp;
      temp = "The X position of the mouse
        is at "
      + window.event.x;
      temp = temp + ". The Y position is
        " +
       + window.event.y + ".";
       theText.value = temp;
    }

  -->
  </SCRIPT>

</HEAD>
<BODY>

<H1>
  This page will always tell you in the
text box below where
  the mouse pointer is. Move the mouse
around and see.
</H1>

<BR><BR>

<TextArea ID = theText Rows=4 COL=40>
</TextArea>

</BODY>
</HTML>
```

Conclusion

I've only touched the surface of what you can do when you take full advantage of the rich event handling available with the new document object model. For example, it's possible stop events at any level from "bubbling up" from a specific element to its container (like a frame) to (eventually) the document itself.

To see more of what you can do in either Navigator 4 and IE 4, I suggest checking out the books "Instant Netscape Dynamic HTML" and "Instant IE4 Dynamic HTML", both by Alex Homer and Chris Ullman and published by Wrox Press. You may recall from the first *View Source* column ("JavaScript History Lesson", *Java Pro*, February/March 1998) that you can use JavaScript code to tell you which browser the script is currently running on when the page loads. This means that when combined with the information contained in good reference books like the ones I just mentioned, you *can* take full advantage of the features available to you in that browser. Finally, to find the full specification for the Word Wide Web Consortium's "Document Object Model," check out http://www.w3.org/DOM/. ■

Property	Description
AltKey	Returns true if the Alt key was down when the event occurred
CtrlKey	Returns true if the Ctrl key was down when the event occurred
ShiftKey	Returns true if the Shift key was down when the event occurred
Button	Reports which mouse button (if any) was down when the event occurred, as follows: 0 = no button, 1 = Left button,2 = Right button, 4 = Middle button
ScreenX, ScreenY	Returns the location of the mouse pointer relative to the whole screen
x, y	Returns the location of the mouse pointer relative to the useable area of the browser (excluding things like toolbars, scrollbars, and the title bar)
KeyCode	Returns the ASCII code of the key that was pressed

Table 3. Event Object Properties. Internet Explorer 4 and the W3 Document Object Model have some properties in common.

Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

Handling Events in JavaScript
Locator+ Code: JP0898VS

Advertiser Index

AccuSoft 1-800-741-7769 http://www.accusoft.com	31	**Mobile Application Servers, Inc.** 1-888-MOAPPS1 http://www.mobileapps.com	4
DevX-Enterprise-Zone 1-800-848-5523 http://www.enterprise-zone.com	70	**NT Enterprise Developer's Conference** 1-800-848-5523 http://www.vbits98.com	40 & 41
DevX-Java-Zone 1-800-848-5523 http://www.java-zone.com	55	**NuMega Technologies** 1-888-686-3438 http://www.numega.com	9
DevX-Premier Club 1-800-848-5523 http://www.premier-club.com	50	**ObjectShare** 1-800-973-4777 http//www.objectshare.com	27
Distinct Corporation http://www.distinct.com	15	**ObjectSpace, Inc.** 1-800-OBJECT1 http://www.objectspace.com	11
Enterprise Development 1-800-848-5523 http://www.enterprisedev.com	73	**Progress Software** 1-800-477-6473 x 4700 http://www.progress.com	21
Inquiry.com 1-800-848-5523 http://www.inquiry.com	63	**ProtoView Development Corporation** 1-800-231-8588 http://www.protoview.com/java	C4
Java Pro CD 1-800-848-5523 http://www.devx.com	59	**Rogue Wave Software** 1-800-487-3217 http://www.roguewave.com	2
Java Pro Magazine Subscriptions 1-800-848-5523 http://www.devx.com	16	**SIGS Conferences** http://www.javadevcon.com	35
KeyStone Learning Systems Corporation 1-800-349-5761 http://www.klscorp.com/jpm	13	**Symantec** 1-888-438-1797 http://cafe.symantec.com	6
KL Group 1-800-663-4723 http://www.klg.com	C2 & 23	**Visual Components** 1-800-884-8665 http://www.F1J.com	C3
Lotus Corporation 1-800-872-3387	1	**Web Builder CD** 1-800-848-5523 http://www.devx.com	62
MindQ Publishing 1-800-646-3008 http://www.mindq.com	53	**WEB Builder Subscriptions** 1-800-848-5523 http://www.devx.com	45

Microsoft's Jview JIT

When it comes to choosing a Java sports car, do you prefer a Porsche or a Ferrari?

by Paul Tyma

Have you ever shopped for office furniture? For some odd reason, office furniture salespeople have adopted a vehicular analogy that isn't always as descriptive as they seem to think. They like to describe different brands of desks in terms of brands of cars; this one's a Pontiac, that one is a Chevrolet. They look at you as if the subtle, luxurious nuances involved in sitting behind an Oldsmobile desk were plainly obvious over sitting behind a Pontiac. They never use exotic or elaborate car metaphors, which might give me some clue (I never happen to sit behind a Ferrari or Porsche, for example). Then again, I guess exotic and speed don't work well as adjectives to describe furniture.

Fortunately, such dramatic descriptions *can* be applied to computers and specifically to Java runtimes. We started with the JDK (probably something like a Dodge) and have quickly moved up to the big boys. Last issue we looked at Symantec Corporation's Version 3.0 Just-in-Time compiler. Definitely a Ferrari if I ever saw one. For this issue we've tested that runtime's arch rival, Microsoft Corporation's Jview JIT. It could be described as a Porsche. Both Symantec's JIT and Jview are fast and exotic.

Microsoft's Jview JIT

Performance
- Blazing
- **Fast**
- Moderate
- Slow

Purity
- **Superset**
- 100%
- Subset
- Anomalies

Runs on: Win95/NT

Execution method: Just-In-Time Compilation

Garbage Collector: Stop and Copy

Availability: Included in IE 4.0, Visual J++, Microsoft Java SDK

Paul Tyma is president and chief scientist at preEmptive Solutions Inc., a Java training and consulting firm. Paul is the lead author of the book *Java Primer Plus* and a frequent conference speaker on Java performance. He is currently a Ph.D. candidate in computer engineering at Syracuse University and lead architect for preEmptive's DashO Java bytecode optimizer/obfuscator. Reach him at ptyma@preemptive.com.

I'm always amazed when I observe two products with different origins, designs, and methodologies that each competitively solve the same problem. This applies to cars, aircraft, and of course, JITs. Different engines, performance characteristics, and general "feels", but each does its job well. All we can do here is outline the idiosyncrasies. In many ways, the underlying design of Symantec's and Microsoft's JITs is radically different, yet they attain comparable and impressive scores.

There is more to today's JITs than the simple idea of conversion between Java bytecode (i.e., class files) and the machine code of a given architecture. JITs no longer rely on the execution of "real" machine code to produce significant performance boosts. JIT compilers do as much code optimization as they can; the implication being that they certainly can't do everything a typical compiler optimizer can do because they don't have the time (JITs do their optimization on-the-fly in front of a waiting user, whereas compiler optimizers have all the time in the world).

Exactly what optimizations are being utilized is hard to tell in any given JIT, but things like simple code motion, dead-code elimination, conservative method inlining, and constant propagation come to mind as likely candidates.

Garbage Collection

The Microsoft runtime has long boasted a "superior" garbage collection scheme. Unlike the JDK's "mark-and-sweep" algorithm, the Microsoft runtime uses a "stop-and-copy" algorithm. Mark-and-sweep tends to produce fragmented memory spaces as it collects garbage. In contrast, the stop-and-copy algorithm copies all good objects from one heap space to another. In fact, this type of collector is often called a "scavenger"

Benchmark Results

Once again, our standard benchmarks were run through the VM to match it up against the competition. In general, the Microsoft JIT and the Symantec JIT were neck and neck. I'm sure you've heard time and time again, how you should be suspicious of so-called "benchmarks". There's no question that benchmarks help portray how a runtime's (or CPU's) performance characteristics will shape up, but it's also true they never tell the whole story. The Microsoft JIT's new approach to garbage collection brought up some further implications in our testing and the results are duly posted. Unfortunately, it's difficult to say whether we're always comparing apples to apples.

Matrix Multiplication. Here are the results (see Table 1). As always, each test was run five times on the same machine and averaged.

The matrix test reinforced our confidence in JITs over interpreters. Remember, the "Total" category is a timing from C: prompt to C: prompt. The hope is that it gives us some idea as to the time spent loading overhead (including code optimization). This benchmark is ripe with optimization opportunities and it looks like both Symantec and Microsoft took advantage of them. Besides Microsoft being slightly faster off the mark, the two JITs ran very close (and both pounded the poor JDK).

	Loop Running Time	Total Running Time
JDK	2020	3020
Symantec JIT	280	1810
Microsoft JIT	270	1500

Table 1. Matrix Multiplication. Time in milliseconds.

Memory. Memory, memory, memory. It's a well-known fact that memory allocation can be painfully expensive, and both JITs received excellent scores. It becomes more difficult to discern exact categories, because we can't force the memory subsystem to do exactly as we say. For instance, it's possible the garbage collector in the Microsoft JIT asynchronously jumped in and ran during the Creation phase, tainting the results. This could account for a slightly worse score during the Creation phase and a better one in the GC phase. Overall, it's best to consider the three scores in unison—only then are we sure that they all did the same amount of work.

Microsoft's stop-and-copy implementation should be faster at allocating memory because it never has to find a suitable chunk of memory because it is devoid of fragmentation problems. However, it could also be slower at general garbage collection, except when the mark-and-sweep algorithm needs to defrag (see Table 2). The test backs up the general impression I've

Continued on next page.

	Creation	Method-Call	GC
JDK	46740	160	440
Symantec JIT	610	50	660
Microsoft JIT	660	50	60

Table 2. Memory. Time in milliseconds.

since it doesn't so much collect garbage as it does rescue in-use objects.

Mark-and-sweep starts with a traversal of all live objects (or pointers) from a canonical JVM object. In other words, if an object is pointed via a set of references starting from the JVM, it is live. Each object encountered is marked as live. When that phase is over, all unmarked objects are assumed dead and collected. Of course, collection typically involves some steps including finalization and memory liberation. But in the final result, there are holes in the contiguous heap where dead objects have been collected. Further memory allocation may be forced to find a suitable memory hole to fit a new object. Eventually, mark-and-sweep algorithms need to defrag (or compact) the heap to consolidate free space at the end of the heap.

Microsoft's stop-and-copy works somewhat differently in that it uses two (usually equally sized) heaps. It allocates continuously until the active heap is full. When the garbage collector runs, it copies all live objects from the current heap contiguously into the other heap. This leaves the dead objects behind and implicitly defragments the heap. It then uses the copied-into heap as the active heap until the next run of the collector.

Purity

The original idea behind a pure runtime was conformance to the Java specification. This runtime seems to straddle both sides of the specification. After spending many hours attempting to appease Microsoft's verifier, I can honestly say it is the most rigorous enforcer of Java's rules. In some cases, the level of security was astounding. I created situations that could only occur through direct bytecode/class-file manipulation, not simple compilation. Microsoft caught them and handily wrestled them to the floor.

Microsoft's verifer is the most rigorous enforcer of Java's rules

On the other hand, Microsoft has added the controversial "windows extensions". I'll not go deeply into those here, but suffice it to say they create non-platform independent Java. To be fair, any seasoned Java programmer knows write-once, run-anywhere across Macs and PCs is a myth. Getting a Java application to run across both takes a lot of specialized work-arounds. Overall, I favor a platform-independent-as-possible approach, but I must admit, my customers were pretty happy with their Windows-specific Java application having that little icon in the system tray.

Conclusion

So, which would you rather have, the Ferrari or Porsche?

Continued from previous page.

gotten from correspondents: the Microsoft VM is superior for general purpose use.

APIs. Our API benchmark, shown in Table 3, puts and gets thousands of objects in the `java.util.Vector` and `java.util.Hashtable`. It again needs a small qualification. The Microsoft VM was running with its own classes.zip, therefore some "system" level classes may run differently. For the first time, we see a solid distinction between the two JITs. Overall, the Microsoft JIT outdistanced Symantec's.

	Putting	Getting
JDK	11810	2300
Symantec JIT	5540	330
Microsoft JIT	2260	440

Table 3. APIs. Time in milliseconds.

It would seem that Microsoft's JIT tends to hit the finish line ahead of Symantec's, but only by a nose. The story is far from over and it will be interesting to see where these two products go from here.

Not a bad choice. Overall, the two JITs perform comparably well. However, keep in mind that Internet Explorer 4.0 is running this new JIT whereas Netscape 4 is still running Symantec's JIT 2.0, which is not in the same league. Love or hate Microsoft (don't we all), the speed and clean design of its runtime is hard to argue with. It is, by a smidge, the fastest runtime we've tested so far.

Microsoft's runtime supports all its platform-dependent additions. In the purest form, this doesn't interfere with your normal Java usage but it's there if you need it. And if you are developing a mostly Windows application, it might be worth your while.

The future of JITs is somewhat uncertain. Microsoft is ensuring the survival of its JIT with unique support of the Windows extensions. If you want to use it, you know where you have to go today. Sun's HotSpot runtime makes a lot of promises on speed and could very well deliver, challenging the lead set by the JITs. We'll be sure to give you the lowdown in a future installment.

The future of JITs is somewhat uncertain

Code Online

All the code in this article is available for free at www.java-pro.com. Use the Locator+ code shown below to download the code. Free online registration is required.

Microsoft's Jview JIT
Locator+ Code: JP0898RT

Doing Objects in Microsoft Visual Basic 5.0
ZD PRESS
JAVA HOW-TO
WINDOWS SERVER
Dan Appleman's
DEVELOPING ActiveX COMPONENTS WITH VISUAL BASIC 5.0
Hitchhiker's Guide to Visual Basic SQL Server
William R. Vaughn
Inside Visual C++
David J. Kruglinski
teach yourself Active Web Database Programming IN 21 DAYS

Save 20% on the Best Programming Books

Order online at DEVX MarketPlace

Buy any book and save 20% when you order through DevX's online bookstore. We carry the latest titles on:

- Visual Basic
- Visual C++
- Java
- ActiveX
- Enterprise development
- and more!

Plus, get 40% off our Featured Books of the Week. Select from our top titles for great savings—but act quickly because each week means another set of titles at incredible discounts!

www.devx.com

FTP Books FAWCETTE TECHNICAL PUBLICATIONS
209 Hamilton Avenue • Palo Alto, CA 94301
Phone: 650-833-7100 • Fax: 650-853-0230

Java is a trademark of Sun Microsystems. Microsoft, Visual Basic, and Visual C++ are registered trademarks of Microsoft Corporation. The Development Exchange and DevX are trademarks of Fawcette Technical Publications.

Spreadsheets Everywhere.

You can make it happen with Formula One for Java, the Internet Spreadsheet.

Delivering spreadsheet functionality where you need it is no longer a problem with Formula One for Java. Leveraging all the benefits of the Java programming language, Formula One for Java is the perfect platform independent solution for applications or applets that need to be delivered by the Internet or an intranet. Its features include:

- Small download and footprint.
- Robust, fast calculations.
- Read/Write Microsoft Excel workbooks.
- Support for more than 130 worksheet functions.
- More than 400 properties and methods.
- Use it as an applet, JavaBean, or standalone application.

Visit our Web site and download a trial. We'll put Formula One for Java in your hands. You'll put spreadsheets everywhere.

VISUAL COMPONENTS®

800-922-9665

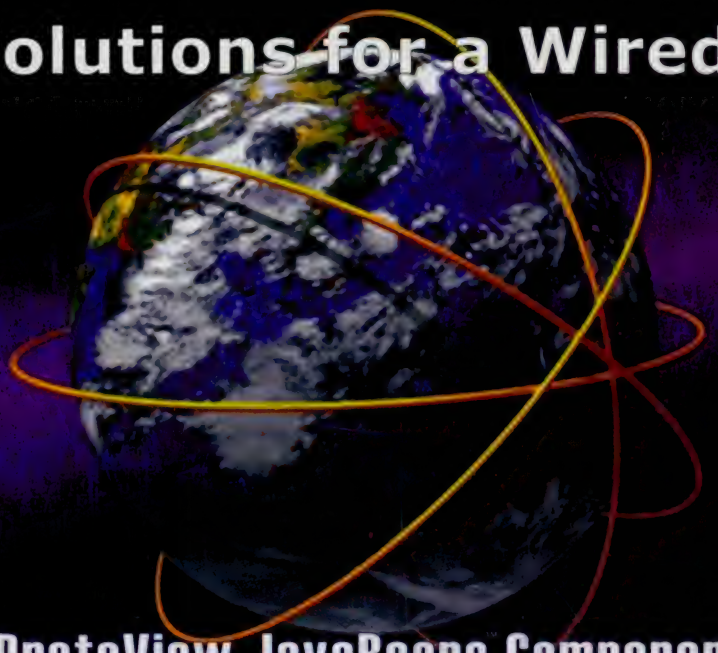
For trials, demos, and information:

www.f1j.com

"The leader in component software since 1993."

©1998 Visual Components, Inc. All rights reserved. Visual Components, Formula One, and Formula One for Java are trademarks or registered trademarks of Visual Components, Inc. All other names are trademarks of their respective companies.

Java Solutions for a Wired World.

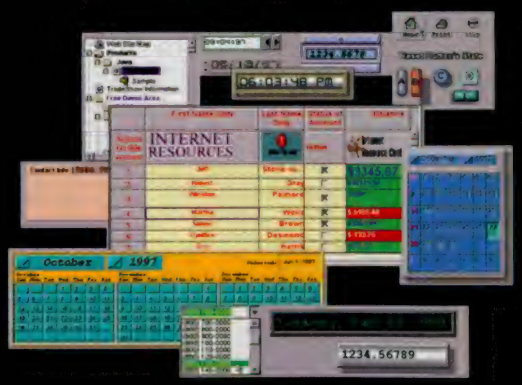


ProtoView JavaBeans Components
Extending JFC With The New Data ExplorerJ™

Data ExplorerJ for JFC™

Use the Windows-explorer metaphor for displaying & editing data in your next Java application. This lightweight component integrates the JFC JTree, JSplitter and JTable into a single integrated component that reduces overall coding.

Data ExplorerJ extends the base classes of JFC and adheres to the Java Component Model and 'Pluggable Look & Feel Architecture' of JFC. In addition to supporting a variety of data models, the right-hand pane can also be customized to display any JavaBeans component or Java panel on a node-by-node basis.



The ProtoView JSuite™

JavaBeans and AWT1.0 classes, includes the DataTableJ grid component with JDBC and Visual Cafe database support. TreeViewJ, CalendarJ, TabJ and WinJ (MaskEdit, Currency, Numeric Edit, DateEdit Long, Time, Combo Box and many more).

Order ProtoView Java Components Online Today!

www.protoview.com

Download FREE Trial Versions of ProtoView Components

1-800-231-8588

PROTOVIEW

Software Tools for Component Based Development

Copyright © 1998 ProtoView Development Corporation. All rights reserved.
All ProtoView product names are trademarks of ProtoView Development Corporation.

Become a Charter Subscriber NOW!
(And You Won't Miss a Single Issue)

JAVA™ Pro

YES ☑ Send me one year (6 issues) of JAVA Pro for just $29.95

Name _____

Address _____

City _____ State _____ Zip _____ Country _____

E-mail address _____

❑ Bill me ❑ Payment enclosed Charge my: ❑ Visa ❑ Master Card ❑ AMEX

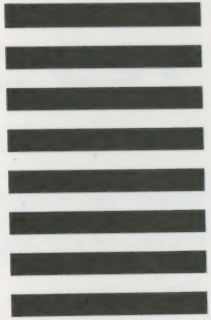
Card # and expiration date _____

Signature _____

One year is 6 issues. Annual cover price is $41.70. Make checks payable to JavaPro. Please allow 6-8 weeks for delivery of first issue. International subscriptions must be payable in U.S. Dollars plus postage. Canada/Mexico add $9/year; all other countries add $22/year for air mail. Canadian GST included.

4007

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 317 Palo Alto, CA

POSTAGE WILL BE PAID BY ADDRESSEE

JAVA™Pro

PO BOX 54584
BOULDER CO 80323-4584

Become a Charter Subscriber NOW!
(And You Won't Miss a Single Issue)

JAVA™ Pro

YES ☑ Send me one year (6 issues) of JAVA Pro for just $29.95

Name _____

Address _____

City _____ State _____ Zip _____ Country _____

E-mail address _____

❑ Bill me ❑ Payment enclosed Charge my: ❑ Visa ❑ Master Card ❑ AMEX

Card # and expiration date _____

Signature _____

One year is 6 issues. Annual cover price is $41.70. Make checks payable to JavaPro. Please allow 6-8 weeks for delivery of first issue. International subscriptions must be payable in U.S. Dollars plus postage. Canada/Mexico add $9/year; all other countries add $22/year for air mail. Canadian GST included. 4007

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 317 Palo Alto, CA

POSTAGE WILL BE PAID BY ADDRESSEE

JAVA™Pro

PO BOX 54584
BOULDER CO 80323-4584